SOCCRATES

SOC & CSIRT Response to Attacks & Threats based on attack defence graphs Evaluation Systems

D4.5 Report on the Threat of Adversarial Examples on AI for Cyber Security

Deliverable type:	Report
Contributing work packages:	WP4 AI based Detection & Prediction
Due date of deliverable:	28/02/2022
Submission date:	28/02/2022
Dissemination level:	PU

Responsible organisation:	AIT
Editor:	Paul Smith
Revision:	1.0

Abstract	This deliverable presents the results from two studies on adversarial machine learning (ML). Their goal is to examine the nature of this risk for cybersecurity-related applications of ML and to highlight the importance of training models in adversarial settings.
Keywords:	Artificial Intelligence, Machine Learning, Adversarial Examples, Genetic Algorithms

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 833481
Call H2020-SU-ICT-2018 • Innovation Action • Start date: September 1st, 2019

Management summary

The SOCCRATES project is developing a novel platform that can be used to support effective Security Operations Centre (SOC) and Computer Security Incident Response Team (CSIRT) activities. In several cases, the components of the platform make use of Artificial Intelligence (AI) and Machine Learning (ML). For example, the AI-based advanced attack detection platform makes use of several intrusion detection systems that leverage ML models, including the DGA Detective tool [1], [2]. Beyond the SOCCRATES project, there is significant industry and research interest in capitalizing on AI and ML to support automation, for example, and implement intrusion detection systems [3], [4]. Consequently, these technologies are playing an ever more important role in defending our critical infrastructures. Set alongside this trend, there is interest in adversarial uses of machine learning, both by threat actors and the research community. Adversarial machine learning can relate to the use of ML to support cyber-attacks and attacks to ML-based systems. There are several types of adversarial machine learning targeted at distinct outcomes (or consequences), including theft of intellectual property and subverting ML model performance.

In this deliverable, two adversarial machine learning studies are presented. The first examines how genetic algorithms can be used to generate Domain Name System (DNS) names, which are well-formed and resemble those created by so-called Domain Generation Algorithms (DGAs), that are not readily detectable by a machine learning model that has been trained on an extensive representative dataset. The goal of this study is to highlight the need for ML models to be trained in such adversarial settings, to improve their robustness to such attacks. The second study examines the potential risk from so-called adversarial example attacks to ML models that are used for Network Intrusion Detection Systems (NIDSs). These attacks have primarily been explored in the image domain [5], [6]; here, we explore the nature of this problem for NIDS and examine whether it is possible to generate attacks that reflect the domain-specific constraints associated with a NIDS. Our theoretical study suggests that NIDS have the potential to be subverted via an adversarial example attack and that further research is needed to understand the operational nature of this risk.

Contributor(s)	Ewa Piatkowska (AIT) Martin Teuffenbach (AIT) Paul Smith (AIT) Fredrik Morten Sætran (MNM) Martin Eian (MNM) Irina Chiscop (TNO) Federico Falconieri (TNO)

Reviewer(s)	Reinder Wolthuis (TNO)

Security Assessment	**See deliverables table for security assessment requirements**
Approval Date	21/02/2022
Remarks	Review performed by Aljosa Pasic (ATOS) with no issues found.

TABLE OF CONTENTS

1 INTRODUCTION – RATIONALE OF THIS DOCUMENT 5
1.1 The SOCCRATES project 5
1.2 This deliverable 6
2 INTRODUCTION TO ADVERSARIAL MACHINE LEARNING 7
2.1 Background and Motivation 7
3 ADVERSARIAL MACHINE LEARNING TO SUPPORT DGA DETECTION EVASION AND MODEL HARDENING 8
3.1 Background and Motivation 8
3.2 Approach 9
3.3 Experiment Approach 9
3.4 Experimental Results 9
3.5 Summary 11
4 CRAFTING ADVERSARIAL EXAMPLES ACCOUNTING FOR DOMAIN-SPECIFIC CONSTRAINTS 12
4.1 Introduction 12
4.2 Adversarial Example Attacks for NIDS Related Work 14
4.3 Subverting Network Intrusion Detection Systems 15
4.4 Evaluation Models 16
4.5 Adversarial Example Attacks against NIDS 17
4.5.1 Feature Analysis and Grouping 17
4.5.2 Crafting Algorithm 18
4.5.3 Attack Budget (Parameters) and Metrics 19
4.6 Experimental Analysis 20
4.6.1 Datasets and Pre-processing 20
4.6.2 Feature Groups 21
4.6.3 NIDS Model Evaluation 21
4.6.4 Attack Evaluation 22
4.6.5 Attacks on Multi-Class Models 24
4.6.6 Attacks on Anomaly-based (Binary) Models 26
4.6.7 Vulnerability Analysis 26
4.6.8 Feature Perturbation Analysis 27
5 CONCLUSION 28
6 REFERENCES 31
7 ABBREVIATIONS 33

1 Introduction – Rationale of this document

This section introduces the SOCCRATES project and defines the goals of this deliverable.

1.1 The SOCCRATES project

SOCCRATES (SOC & CSIRT Response to Attacks & Threats based on attack defence graphs Evaluation Systems) is an EU funded project under the Horizon2020 programme that has the following main challenge:

How can SOC and CSIRT operations effectively improve their capability in detecting and managing response to complex cyber-attacks and emerging threats, in complex and continuously evolving ICT infrastructures while there is a shortage of qualified cybersecurity talent?

The main objective of SOCCRATES is to develop and implement a security automation and decision support platform (‘**the SOCCRATES platform’)** that will significantly improve an organisation’s capability (usually implemented by a SOC and/or CSIRT) to quickly and effectively detect and respond to new cyber threats and ongoing attacks.

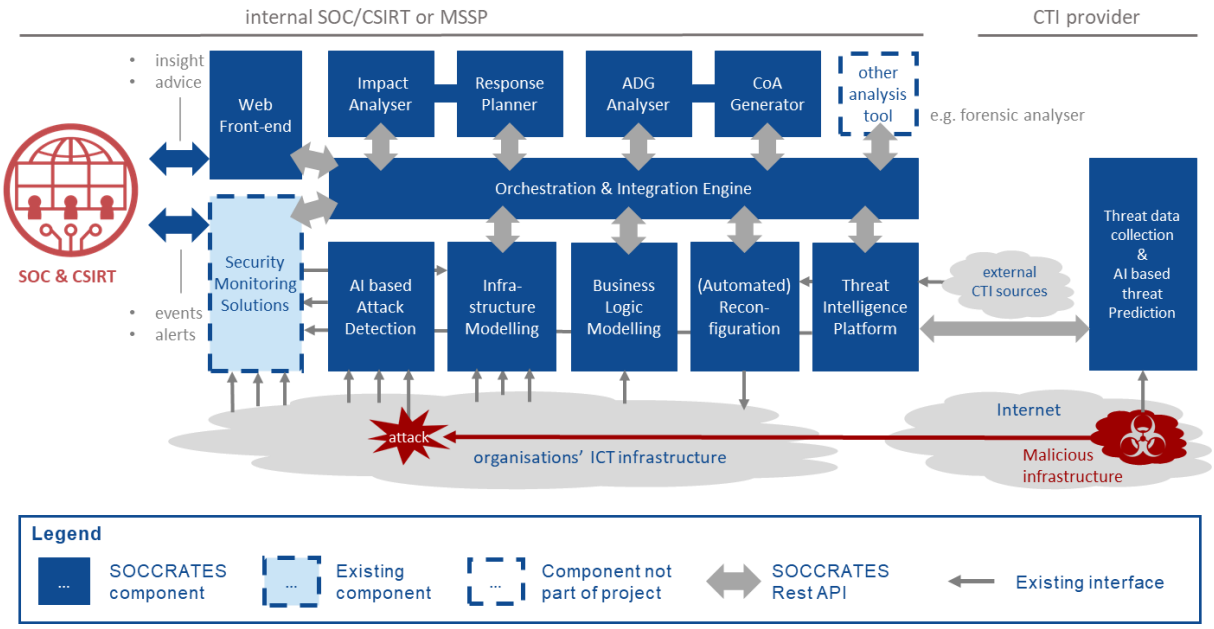


Figure 1.1 – The SOCCRATES platform

The SOCCRATES platform (see **Figure 1.1**) consists of an orchestrating function and a set of innovative components for automated infrastructure modelling, attack detection, cyber threat intelligence utilization, threat trend prediction, and automated analysis using attack defence graphs and business impact modelling to aid human analysis and decision making on response actions, and enable the execution of defensive actions at machine-speed.

SOCCRATES has the following concrete project objectives:

1. Deliver the SOCCRATES platform consisting of an orchestration function and a unique integration of innovative background solutions that seamlessly work together.
2. Show that the SOCCRATES platform can improve SOC operations by evaluating the SOCCRATES platform in two diverse real-life pilot environments.
3. Examine and illustrate the benefits of automation for selected SOC activities to help manage the cyber security skills gap in organizations.
4. Prepare for successful exploitation by the SOCCRATES partners of the individual innovated components and the integrated SOCCRATES platform in commercial products that are offered to the market and are available for the European (business) community.

Please visit www.soccrates.eu for more information on the SOCCRATES project.

1.2 This deliverable

The SOCCRATES project is concerned with improving the effectiveness of SOC and CSIRT activities via the platform that is being developed in the project. Several parts of this platform make use of Artificial Intelligence (AI) and Machine Learning (ML). Perhaps, most notably, the AI-based attack detection capability leverages several machine learning models, including those that are based on artificial neural networks, to detect cyber-attacks. These outcomes are aligned with a wider trend to leverage these technologies to, for example, support automation and detect and classify malicious behaviour. In short, AI and ML are becoming increasingly critical to the cybersecurity of enterprises and critical infrastructures.

Alongside this trend, there is interest in adversarial uses of ML. This can take many forms and includes the malicious use of ML to support cyber-attacks, e.g. to make them more effective, and attacks to machine learning-based models and systems. This latter issue can take many forms, including evading models, model poisoning, stealing intellectual property that ML models encode, and causing a Denial of Service (DoS). Arguably, our understanding of this threat is still in its infancy – in many cases, threats have been explored theoretically and, most often, for the image classification domain (where ML innovations often originate).

It is in this setting that this deliverable explores adversarial machine learning in the context of attacks to ML models and applications for cybersecurity. We focus on two forms of adversarial machine learning – the use of genetic algorithms to generate data that can evade detection by a model, and the generation of adversarial examples that are intended to result in the misclassification of test data (such as that associated with attacks). Studies of this nature are important to understand the nature of this threat. Moreover, they provide insights into how ML models can be hardened against these forms of adversarial behaviour, thus ensuring that our cybersecurity solutions that make use of ML (such as the SOCCRATES AI-based attack detection capability) are less vulnerable to these forms of attack.

1.3 Structure of this deliverable

This deliverable is structured, as follows: in Section 2, we present the motivation for this work, introducing the concept of adversarial machine learning and the two studies that we have conducted in the SOCCRATES project. The first study – presented in Section 3 – examines the use of genetic algorithms to generate Domain Name System (DNS) names that have similar properties to those created by Domain Generation Algorithms (DGAs) that are not readily detectable by a model that was develop in the SOCCRATES project. This study has demonstrated the need for model hardening using such adversarial machine learning techniques. Recently, the topic of so-called *adversarial examples* has been attracting a significant amount of attention from the research community. Adversarial examples are images that

are intentionally and imperceptibly – to the human eye – manipulated to cause misclassification. In Section 4, we present the results from a study of this form of adversarial machine learning for Network Intrusion Detections Systems (NIDSs) – in this setting, instead of images, the test data are communication network flows. Finally, in Section 5, we present some conclusions and highlight areas for future work on this important topic.

2 Introduction to Adversarial Machine Learning

2.1 Background and Motivation

Artificial Intelligence (AI) and machine learning (ML) have found application in numerous domains and applied to tasks that are increasingly critical. This trend has been driven by access to an abundance of so-called big data and a renewed interest in approaches that make use of artificial neural network (ANN) models and deep learning. The cybersecurity sector is not impervious to this interest in the application of AI and ML. Perhaps the most common application of ML in this context is to support the detection and classification of cyber-attacks, e.g. as part of antivirus or intrusion detection systems. As an example, the SOCCRATES project is developing a so-called advanced AI-based attack detection capability [2] that aims to combine different machine learning approaches to effectively detect advanced persistent threats. This includes an approach to detecting malware-generated Domain Name System (DNS) names, leveraging data that is available from the Shadowserver Foundation [1].

Set against this increasing use – and dependence – on machine learning, adversarial attacks to and applications of machine learning have begun to be observed "in the wild" and examined by the research community. A reflection of this trend can be seen in the creation of the MITRE Corporation's Adversarial Threat Landscape for Artificial-Intelligence Systems (ATLAS)[1] framework – an organization of adversary tactics and techniques associated with machine learning, in a similar fashion to the widely-known ATT&CK framework[2]. Adversarial behaviour in relation to machine learning can be broadly organized into two categories: (i) the use of this technology to support cyber-attacks, in general; and (ii) attacks to machine learning-based models and systems, in order to, e.g. subvert their utility or steal intellectual property (this is associated with a model). An example of the former is the use of machine learning to generate more effective phishing email messages [7]. Furthermore, machine learning algorithms can be used to attack and subvert the functioning of a target model. In this deliverable, we explore two such forms of attack – the use of genetic algorithms to generate malicious data (in this case, Domain Name System (DNS) names) that are not detected by a trained model, and approaches to generating so-called *adversarial examples*, whose purpose is to subvert a target model. A motivation for conducting these analyses to is to "harden" the machine models to these forms attack and, in some cases, improve a model's performance if they are trained in such an adversarial setting [8].

Arguably, we are still at a relatively early stage of understanding the nature of this emerging threat. Whilst there have been some real-world cases of the use of adversarial machine learning, a great deal of the activity that is associated with this topic is being explored by the research community. Much of this work explores its theoretical foundations, e.g. exploring the susceptibility of models to adversarial examples, and does not consider their practical use by an adversary, including their motivation for doing so and the capabilities they would require. The MITRE ATLAS framework represents foundational

[1] https://atlas.mitre.org/
[2] https://attack.mitre.org/

work in this regard, defining an approach to systematize this knowledge, potentially leading to actionable insights that can support organizations to defend against this emerging threat.

In this deliverable, we summarize the findings from two studies that explore two forms of adversarial machine learning that have the potential to subvert the utility of detection capabilities for cybersecurity. The first is intended to demonstrate and illustrate the importance of model hardening to adversarial machine learning techniques with immediate practical benefits. In this work, we explore the use of genetic algorithms to generate test data – in this case, maliciously-generated DNS names – that are not readily detectable by a model that is trained on a very large dataset. The second study focuses on the emerging challenge of so-called *adversarial examples* – test data that is intentionally manipulated to cause a model to misclassify it. Adversarial examples have been explored primarily for the image classification domain, wherein the problem is defined in terms of manipulating images, such that they are misclassified by a model, without the changes being perceptible to the human eye. In this work, we explore this problem for machine learning models that are used for intrusion detection systems, exploring what constraints – such as those related to imperceptibility – are relevant in this context. We show that successful adversarial examples can be generated for network intrusion detection systems (NIDS) that perform single and multi-class classification. However, we are still at a relatively early stage of understanding the risk associated with this form of threat, e.g. regarding the motivation for an adversary to conduct such an attack and the nature of the capability required to do so.

3 Adversarial Machine Learning to Support DGA Detection Evasion and Model Hardening

3.1 Background and Motivation

In the SOCCRATES project, an approach has been developed to detecting and classifying Domain Name System (DNS) names that are generated by malware using Domain Generation Algorithms (DGAs) [1]. Since the DGA classifier is supposed to be deployed in an adversarial environment, it is useful to run a series of attacks against the trained model to find ways to bypass it. This is useful for threat modelling, so that weak spots can be found and fixed, which helps in hardening the model. Additionally, this may help in understanding what featureless models are learning. Featureless models hold the promise of automating the process of feature engineering to learn meaningful feature representations by operating on the raw input signals and being exposed to a large volume of training data. This is a powerful technique but can be a double-edged sword: on one hand it requires substantially less work than feature engineering, but it is not immediately apparent what the model is learning.

Genetic algorithms help search the input space and find the important properties that push the input over the decision boundary of the model, which can be used in determining what are important traits for classification. Results using this method can help reveal surprising properties of featureless models, which may not have been obvious using commonly used performance metrics. Domains that are generated using a genetic algorithm can also be used to improve detection rates by adversarially training on the samples that bypass the model. By trying the same process after performing a training run on different subsets of training data, the previous approaches that uses the genetic algorithms should not work (i.e. the model has been hardened to the DNS names that have been created by the genetic algorithm). This approach differs from other published approaches, focusing on Generative Adversarial Networks (GANs), in that it searches the space outside the training data.

3.2 Approach

Genetic algorithms [9] are a general framework that seeks to systematically search the space of solutions, evaluating the solutions using a fitness function. In this work, we modelled the DGAs as individuals in a population, which go through a series of generations until a sample is below a classification threshold. The model used is DGA Detective – the detector developed in the SOCCRATES project [1] – and was used as an oracle, wherein the model output from the logistic function can be interpreted as a maliciousness score, a score close to 1 indicates a DGA generated domain. The solutions were optimized based on the prediction score. The goal is to minimize this score until it is below the threshold.

The procedure starts by creating random domains and appending a random Top-Level Domain (TLD), e.g. *.com* or *.org*. Every iteration consists of the following steps:

1. **Reproduction:** This process identifies the individuals (domain names) that minimize the maliciousness score, as much as possible. It mates the most promising individuals and create a new generation where the individuals are tried again.
2. **Crossover:** Each successor is generated from two subjects from the current population; a successor is created with an appropriate combination of randomly selected parts of gene strings from both parents.
3. **Mutation:** Here genes are inserted into the successors randomly, so the solution does not get stuck in a local optimum. The possible mutations are all the characters, which are valid input for the classifier.

In this work, we set constraints on the procedure such that the generated domains should not be easily detected by traditional approaches [10]–[12]. They should also adhere to the IETF Request For Comments (RFCs) that pertains to domain names [13], so that they are valid domains. They should still look like DGAs, but are not detected by the classifier, and should have raised suspicion if reviewed by a human analyst.

3.3 Experiment Approach

The experiments were performed on DGA Detective, which is a model trained to classify DGA domains from sandbox runs in the Shadowserver Foundation's sandbox. The model is based on Temporal Convolutional neural networks [14], which is trained on a dataset of 1,244,715 DGA domains derived from DNS queries acquired during execution of malware in the sandbox. Additionally, approximately 2,000,000 benign domains were sourced for training. This fully trained model was made available for download[3], providing binary predictions of domains. To perform these experiments, the model was downloaded, and the code had to be changed to return the result of the logistic function to acquire the probability, indicating the confidence of the model in the domain being a DGA generated domain. The GA could then query the model locally and receive input on how to best optimize the inputs, using the crossover and mutation operations.

3.4 Experimental Results

Figure 2 shows the unigram distribution of the domains generated by the genetic algorithm. The dash ("-") shown in the first column stands out in that it is a deciding factor for the classifier in determining if the domain is malicious. This can probably be attributed to them not being present in the training set of the DGA-generated domains. This creates a blind spot, which can be exploited – domain names that include this character are likely to be not detectable by the algorithm.

[3] https://gitlab.com/cossas/dgad

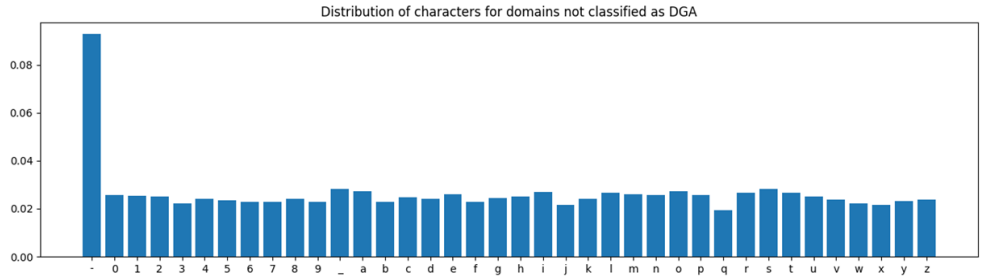


Figure 2: Unigram distribution of the domains generated by the genetic algorithm

Table 1 shows some of the domains that are generated using genetic algorithms and the corresponding scores returned by the model. Compared to the scores that are generated by domain names that are classified as DGA-generated by the model when using the training set that is provided by the Shadowserver Foundation [1], these scores are very low, suggesting they would not be detected by the classifier.

Table 1: Example domain names generated by the genetic algorithm and their classifier score

Domain	Score
60b4_5-x_aqiptw.org	1.866341153800022e-05
d6s80jhvyo1h-4-.pl	1.4575764907931443e-05
g2-2pkn-zlpbnay.nl	3.623345401138067e-05
-r-v4fiy3kd69do.xyz	2.1402278434834443e-05

Figure 3 shows the unigram distribution of domains that are classified as a DGA while trying to bypass the model using genetic algorithms. It shows that the characters are uniformly distributed, but most of them do not contain the dash. This is probably because not a lot of DGA-generated domains contain the character, but it is still a valid domain name.

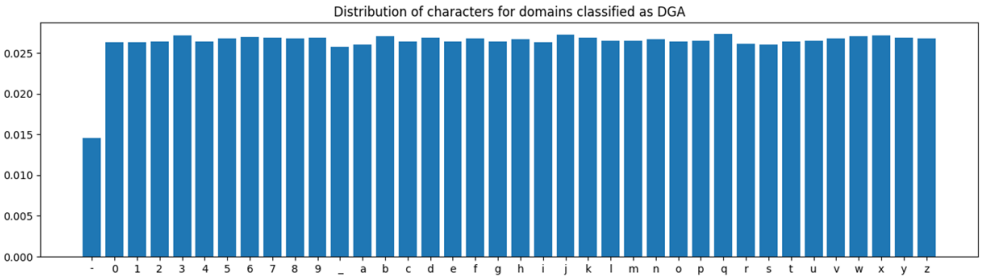


Figure 3: The unigram distribution of DNS names, which are generated using the genetic algorithm, that are classified as DGAs

As a last experiment, the dash is removed from the initialization and the set of possible mutations, which makes it harder to create evasive domains and helps determine whether there are more weak spots. Figure 4 shows the unigram distribution of domains that evade the classifier using the genetic algorithm without "-" in the mutations. These unigram distributions start to resemble the Alexa top one million distribution (see Figure 5).

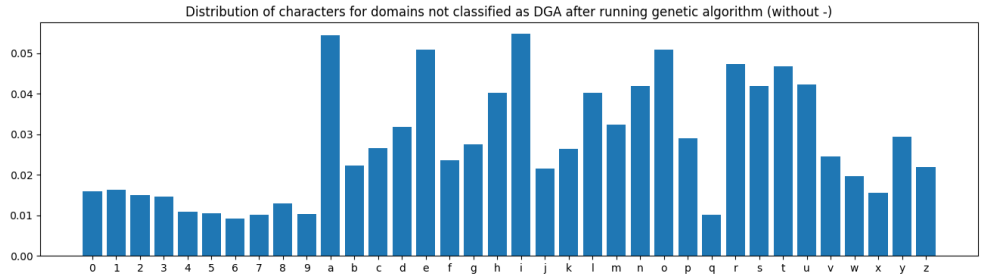


Figure 4: The unigram distribution of DNS names when generated using the genetic algorithm, without the use of dashes

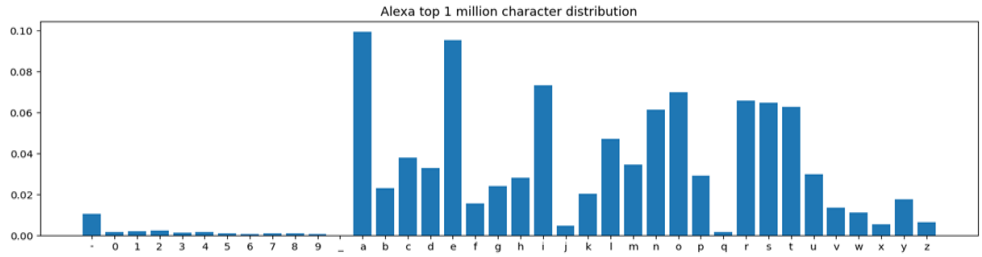


Figure 5: Unigram distribution of domains names from the Alexa top one million domains

Table 2 shows some of the domains that are generated using the genetic algorithm while leaving out the dash in the initialization and mutation steps. The algorithms are still able to get a low score, indicating that the classifier would not be able to readily detect these domain names.

Table 2: Example domain names and their classifier scores for names that are created without a dash

Domain	Score
riuro693trukoik.no	0.019893288612365723
yuncrobizypyerf.com	0.03488051891326904
92tnamsoovonlra.xyz	0.056822001934051514
iphupenbpert20z.nl	0.03501611948013306

3.5 Summary

In this study, we have explored the use of genetic algorithms for creating domain names that are not readily detectable by a model that is trained on a large dataset of known DGA domain names. These results highlight the importance of hardening machine learning models to this form of subversion before they are deployed. Moreover, we suggest that models should be periodically retrained in such an adversarial way, to ensure their performance is not compromised, as adversaries change their procedures (e.g. modify DGAs) and use such forms of adversarial machine learning themselves to identify model weaknesses.

4 Crafting Adversarial Examples Accounting for Domain-Specific Constraints

4.1 Introduction

Adversarial Examples were first discussed by Szegedy *et al.* [5] where they are referred to as *blind-spots of neural networks*. Later, Ian Goodfellow *et al.* named and defined the phenomenon and provided the following definition [15]:

Adversarial examples are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake.

The goal of an adversarial example is to get a classifier to misclassify an instance by adding a small perturbation to the input. Figure 6 shows an example presented by Goodfellow.

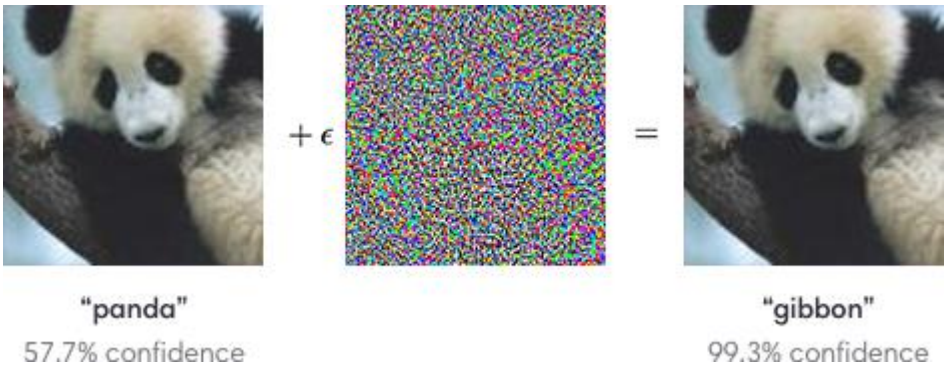


Figure 6: Adversarial Example [15]

A small distortion (of size ε in Figure 6) causes the classifier to change the originally correct prediction. An important aspect of this is that a human observer is not able to distinguish the adversarial example from the original image. This idea is often referred to as *imperceptibility* for a human observer and is almost always implied when adversarial examples in the computer vision domain are investigated.

Formally, for a classifier $F: x \mapsto y$, where $\mathrm{x} \in \upsilon$ sample-space and $\mathrm{y} \in \gamma$ set of labels, and adversarial example x' is defined as:

$$\mathrm{x}' = \mathrm{x} + \delta \; \mathbf{s.t} \; \mathrm{F}(\mathrm{x}) \neq \mathrm{F}(\mathrm{x}')$$

In practice this equation is often formulated as an optimization problem, where one solves for the minimal distance δ that fulfils $\mathrm{F}(\mathrm{x}) \neq \mathrm{F}(\mathrm{x} + \delta)$.

As mentioned earlier, a classification model is a multidimensional function $\mathrm{F}: \mathrm{x} \rightarrow \mathrm{y}$ where X is the input vector and Y the output vector. In the case of anomaly detection, Y would be a scalar indicating whether the input is anomalous. The objective of an adversary is to find the optimal perturbation δ. Researchers have developed optimization approaches to this objective; the most prominent algorithms are the *Fast Gradient Sign Method (FGSM)* [15], *Jacobian based Saliency Map Attack (JSMA)* [16], *Deepfool* [17] and the *Carlini and Wagner L2 norm Attack (C&W)* [18]. A review of proposed adversarial examples crafting algorithms can be found in [19].

Threat models have been defined for adversarial examples, which describe an adversary's goal and the constraints under which they can generate examples. The following taxonomy is taken from Papernot *et al.* [16] and Carlini *et al.* [20]. Papernot defined the threat model of an adversarial attack by defining an *adversarial goal* and *adversarial capabilities*.

The adversarial goal is to impact the integrity of a model's output. The four primary goals of an adversary, enumerated by increasing difficulty to realize, are summarized in Table 3:

Table 3: A summary of four goals related to adversarial examples

Level	Description
1	**Confidence reduction:** Reduce the output confidence of classification. This goal does not require the model to misclassify, but only to increase the uncertainty. This could be problematic if the confidence measure is used to support classification or use of the results from a model.
2	**Misclassification/Non-targeted attacks:** Change the output classification to *any* class that is different from the original class.
3	**Targeted misclassification:** Produce inputs (generate new input) that are classified as a specific target class. The idea is not to add a perturbation to a known instance, but to generate an instance of class that is misclassified. For example, to have attacks classified specifically as benign by a model when they would otherwise be found malicious.
4	**Source/Target misclassification:** Add perturbation to an existing input to force the classifier to classify it as a specific target class.

The adversarial capabilities can be organized into five levels, as shown in Table 4:

Table 4: Five levels of capabilities for adversarial examples

Level	Description
White-box Attacks	
1	**Training data and network architecture:** The adversary has perfect knowledge of the classification model (weights, biases, number of layers) and access to the training data. The attacker can analyse the training data and has full gradient information.
2	**Network architecture:** The adversary has perfect knowledge of the classification model (weights, biases, number of layers) without the training data. With this the adversary has access to the gradient information.
Black-box Attacks	
3	**Training data:** The adversary can collect pairs of input and output data in the training phase. The adversary has no knowledge of the model's architecture but can reproduce the original model using the training data.
4	**Oracle:** The adversary has no information about the original model but can query it. Using various inputs, the adversary can approximate the gradient using a finite difference approach.
5	**Samples:** The adversary can collect pairs of input and output data in the test phase. They cannot modify these inputs.

The first two levels of adversarial capability refer to *white-box attacks* (where knowledge of the network architecture is known) and the latter three black-box attacks (in which information about the network architecture is not known to the adversary). Figure 7 depicts this taxonomy or threat model.

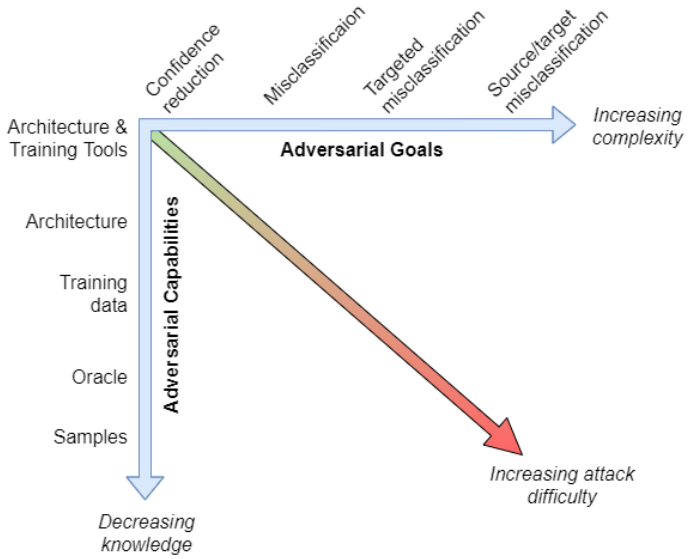


Figure 7: A taxonomy or threat model for adversarial examples from [16]

This taxonomy was developed for adversarial example attacks against models that are used for image classification. To apply it to other domains, Carlini *et al.* [20] extended the concept of adversarial capabilities by splitting it into two subcategories: (i) *adversarial knowledge*, which Papernot *et al.* call adversarial capabilities [16]; and (ii) *adversarial capabilities*. This includes reasonable constraints for the adversarial instances, for example, restricting the size of the perturbation ε to be small or applying constraints – a so-called *perturbation budget* – in the feature space. We use this definition of adversarial capability in this work.

4.2 Adversarial Example Attacks for NIDS Related Work

Most of the research that has been conducted on adversarial examples has been undertaken in the image classification domain [5], [6]. As mention earlier, in our work, we investigate this potential threat in the context of NIDS – an important cybersecurity technology. Initial research that investigated adversarial examples for NIDSs, e.g. the work by Yang *et al.* [21], often considered network traffic datasets arbitrarily – i.e. they do not constrain the perturbations that can be introduced to craft an adversarial example. Consequently, adversarial capabilities were not thoroughly considered for most of the proposed attacks. This is appropriate for attacking an image-based input, but not for network traffic-based input – unlike pixels, which can take any value from 0-255 and be independently changed, features in network flow datasets contain dependencies and have compliance constraints.

Recently, Zhang *et al.* [22] investigated a reinforcement learning approach to match IDS dataset-specific restrictions. The action space of their algorithm only contains valid actions, meaning those actions that would not degrade the validity of the instance. Hashemi *et al.* [23] introduced the idea of treating features differently, based on their properties, when crafting adversarial examples for flow-based NIDSs. They also take dependencies between features into account. We have extended this research by assigning weights to feature groupings, which reflect perturbation constraints, and incorporating these weights into the optimization function that is used to craft the adversarial example.

There are several approaches to measuring the robustness of algorithms against adversarial examples. As the concept of adversarial examples for image recognition algorithms involves imperceptibility from the original image for a human observer, most robustness metrics use a distance metric. This is appropriate in this domain, because the distance between the original image input and adversarial example is correlated with the visible difference. Moosavi *et al.* [17] define the expectation value of the *minimal perturbation* over a test dataset as a measure of robustness. Using a theoretical approach, Weng *et al.* [24] have developed the CLEVER score, which is also an estimation for the minimal distance required to subvert a neural network classifier. Papernot *et al.* [16] have proposed the *adversarial distance*, which measures distances between different labels using gradient information to indicate the risk of misclassification between classes. Meanwhile, in the NIDS domain, Hartl *et al.* [25] have developed the Adversarial Risk Score (ARS), which is a distance-based robustness score for classifiers against adversarial examples. In their work, they use a Recurrent Neural Network (RNN) for classification and investigate the feature sensitivity of their classifier. We argue that without considering the properties of features, e.g. to what extent they can be manipulated, a distance-based approach for measuring the effectiveness of adversarial examples against NIDSs does not accurately reflect the threat. We will elaborate an empirical score for robustness that is based on the relative number of adversarial examples within a given (constrained) feature subspace.

4.3 Subverting Network Intrusion Detection Systems

The widely-accepted definition of adversarial examples requires adaptation for NIDSs. This is because the concept of imperceptibility (to the human eye) of the perturbations does not readily apply in this context. Therefore, we define *adversarial examples for NIDS as instances (e.g. network traffic flows) with anomalous properties that are intentionally designed to cause the model to misclassify them and yet still have their intended purpose.* The point regarding purpose – a form of constraint – invites explanation: as an example, if the network traffic is intended to result in a Denial of Service (DoS) attack; after introducing a perturbation, the attack should have its intended impact of saturating the resources of an intended target.

Based on an adversary's goals and capabilities (see Figure 7), and this definition of adversarial examples, a threat model can be defined, including an attacker's objective. A likely objective of this form of attack is to conduct a stealthy network attack (e.g. a port scan) and not have it detected by a NIDS. In other words, the goal is to generate adversarial examples that are malicious but are misclassified as being benign by a NIDS. In the organization of attacker goals described earlier, this is a form of source/target misclassification.

Figure 8 depicts the approach for generating adversarial example attacks against a target NIDS. An unknown – to the adversary – NIDS is monitoring the network traffic of a target network. By assuming an attacker can monitor the system or to gain access to input-output samples, they are able to reproduce the target NIDS and create a so-called *substitute model*. By crafting adversarial examples for this substitute model, an attacker might be able to bypass the original detection system. For the first part of the evaluation, knowledge about the training samples is assumed. With this approach, the transferability between a black-box and a substitute model will be evaluated [26], [27]. For our final evaluation, the architecture of the target model is known, which is equivalent to assuming the substitute model is a perfect copy of the original system.

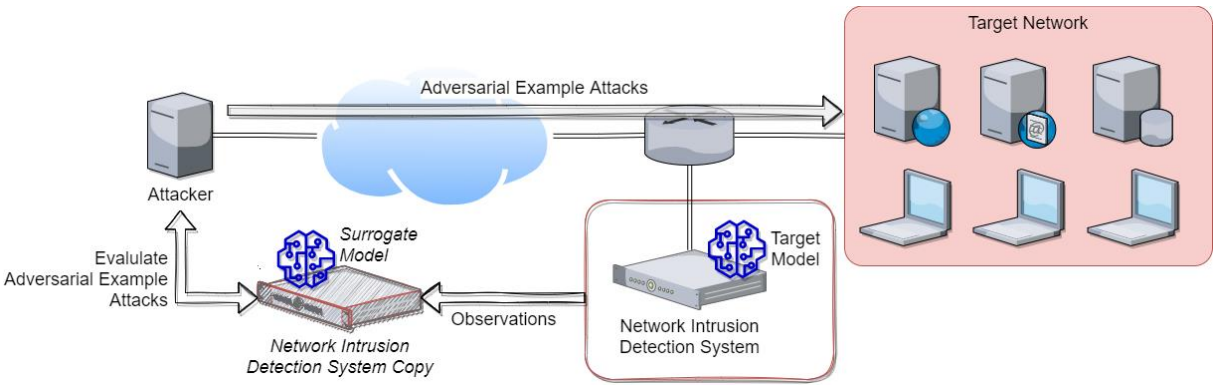


Figure 8: Overview of the target environment for generating adversarial examples for NIDS

An important difference between generating adversarial examples for a NIDS versus a model used for image recognition relates to the adversarial capabilities. Pixels in images can be changed arbitrarily without taking any dependencies between the pixels and similar constraints into account. However, for network data there are such dependencies that must be considered – i.e. it is not possible to change features that describe network flow behaviour in the reverse direction, as they are not controlled by the adversary. To define the adversarial capabilities for the attack scenario, an approach for categorizing and a rule-set for manipulating features of a NIDS dataset is proposed. This contains a restricted feature-space and a maximum perturbation budget. The idea is to restrict the feature-space to feasible features, i.e. features an attacker can change without losing important properties of the attack, like a feature-space reduction technique done by Eykholt *et al.* [28].

4.4 Evaluation Models

To get a representative sample of deep learning algorithms for our experiments, we use several state-of-the-art models: a Deep Neural Network (DNN) for binary and multi-class classification [29], in a supervised and semi-supervised fashion; a deep belief network for multi-class classification (semi-supervised) [30]; and an Autoencoder (AE) that is trained to perform outlier detection for unsupervised anomaly detection [31]. These models are briefly described:

- **Deep Neural Network:** To extend the functionality of a regular DNN, which usually works as a supervised classifier, Rezvy *et al.* [29] proposed to stack an Autoencoder on top of a two-layer classifier network. This algorithm trains the model in three stages: *(i)* the AE is trained unsupervised; *(ii)* the classifier is trained with the output of the AE as input to perform a classification, in a supervised fashion; and *(iii)* the network is tuned with a few training samples, also supervised. We use this DNN to perform binary classification, to get a comparable result to the AE-based outlier detection (see below).
- **Deep Belief Network:** A model with a similar approach to the AE-DNN is a Deep Belief Network, which is a perceptron network that consists of several layers of Restricted Boltzmann Machines (RBMs). RBMs are often used for dimensionality reduction or feature extraction, which is similar to the algorithm proposed by Gao *et al.* [30]. The model consists of several layers of RBMs with a single-layer classifier stacked on top. The RBMs are trained unsupervised, the classifier then supervised, using the output of the final RBM as input. The idea is that the RBMs in sequence reduce the input dimension to achieve a better representation of the features.
- **Autoencoder:** Hawkins *et al.* first proposed the use of an AE to perform outlier detection [31]. The approach is to train an AE on benign data (train the network to reproduce the input, unsupervised) and then use the reconstruction error to measure for outliers. The Outlier Factor

(OF) of a record i is defined as the average of the squared reconstruction error. To get a more general outlier detection, we extended this concept with a method proposed by Azami *et al.*, which converts distance-based outlier detection methods to probabilities using a sigmoid function [32].

4.5 Adversarial Example Attacks against NIDS

As mentioned earlier, there are several domain-specific aspects to be considered when crafting adversarial examples for NIDSs, in contrast to doing so for images:

1. The features extracted from network traffic are heterogeneous, including flows, context, statistical and categorical features. The dependencies between features imposes additional limitations on how their values can be modified.
2. Adversarial perturbations should be crafted considering the way the attack will be implemented, e.g. via the modification of network flows. Therefore, perturbations should render correct flows and ensure compliance with protocols.
3. The concept of imperceptibility, which is applicable in the image domain, does not readily apply in the NIDS domain. Nevertheless, minimising perturbation may help to ensure that the original malicious intent of the network attack is preserved. In addition, restrictions on perturbation magnitude could also be important to remain stealthy, i.e. introducing high perturbation to some features might trigger alerts. For instance, increasing packet numbers to hide one attack could result in it being detected as another, either by a NIDS or complementary anomaly detection algorithms.

In the following, we present how we address these aspects. Based on the properties of the features, we divide them into groups (in a similar manner to [23]) and incorporate feature constraints in the adversarial crafting algorithm. We propose weighted optimisation for crafting adversarial examples, in order to ensure feature constraints. As with previous work, we apply perturbation constraints to remain stealthy.

4.5.1 Feature Analysis and Grouping

Hashemi *et al.* [23] proposed the idea of grouping flow-based features by their *feasibility*, i.e. grouping flows based on whether they can be modified by an adversary and still yield correct flows. We follow a similar approach and group features using the organization presented in Table 5.

Table 5: Organizing network traffic features into groups, based on how readily they can be perturbed, based on [23]

Group	Description	Weight w
G0	Features extracted from backward flows and features whose values depend on the other features and can be calculated directly by a set of them	0
G1	Independent and not used to derive other features	1
G2	Independent and used to derive other features	2
G3	Features dependent on batches of packets (e.g., mean and frequency-based features) and features with underlying physical constraints (e.g., the inter-arrival time of packets)	3

We exclude features that are related to backward flows, as it is reasonable to assume that an adversary generally does not have control over traffic that is generated by other hosts in the network (unless they are compromised). Features that are derived from other features are also not considered. These

two groups are merged into a group (G0). Next, we consider features that are independent, e.g. the total number of forwarded packets or maximum length of the forwarded packets. These are further divided into features that are not used to derive other features (G1) and those which are (G2). Changes to features of the latter group requires recalculating dependent features; therefore, we consider them harder to change and assigned them a higher weight. The last group contains features that are difficult or impossible to change directly. For instance, to influence statistical summaries of the flows requires the modification of multiple packets. This last group (G3) also includes features with underlying physical constraints, whose modification could cause potential violations and break the communication.

Based on this grouping of features, we assign a weight w to each of the groups (as shown in Table 5). The weights give an intuition about how difficult it is for an adversary to perturb a feature, wherein $w = 0$ denotes that the feature cannot be changed and $w = \{1,2,3\}$ indicates increasing difficulty. The crafting algorithm is intended to favour features with $w = 1$ over higher weights. These weights are used in our crafting algorithm. They can be assigned to groups of features or individually to each feature. In this work, we have used constraints that are imposed by the implementation of adversarial perturbation through changes in network flows. Assuming a different threat model, these constraints can differ. For example, weights could also reflect the risk – to the adversary – associated with their change (e.g. due to complementary security measures).

4.5.2 Crafting Algorithm

For crafting adversarial examples, we extend the Carlini and Wagner (C&W) attack [33]. This method is among the most powerful crafting algorithms and a benchmark technique to evaluate the robustness of deep learning algorithms. Carlini and Wagner formulated an optimisation-based approach to craft adversarial examples. They derived an objective function that maximizes the desired target prediction, while minimising the size of the perturbation:

$$\min_{\delta} \left(\left|\left|\delta\right|\right|_p + c \cdot g(x+\delta) \right) \quad \text{s.t. } x+\delta = x^* \in [0,1]^n,$$

where $\delta = x - x^*$ is the distance between the input x and the adversarial sample x^*, c is a coupling constant and $g(x)$ is a target function. This approach ensures minimal perturbation while minimizing the desired target function. For the L_2 (i.e. Euclidean) norm attack, p is set to 2. The constant c in this equation links the minimization of the distance with the minimization of the target function – smaller values of c result in a bias toward minimizing the distance. The optimisation is solved with the Adam algorithm [34].

As this crafting algorithm performs a distance minimization, it is possible to extend it with our proposed weighted features approach. Features are assigned weights w according to their grouping (0 to 3). Using these weights, along with a mask that specifies the set of features that are considered by the algorithm, we extend the C&W attack, as follows:

$$\min_{\delta} \left(\left|\left|\delta \odot w\right|\right|_2 + c \cdot g(x+\delta \odot mask) \right) \quad \text{s.t. } x+\delta = x^* \in [0,1]^n,$$

where $\odot$ indicates an element-wise vector-vector multiplication. The weights added to the distance forces the algorithm to favour modifications on low-weighted features and avoid adding too much perturbation to high-weighted features. The $mask$ represents our restriction on the feature space,

e.g. it sets the perturbation of undesired features to 0 (in our case features from G0). As the function $g(x)$in the proposed approach, we used the f_5 function, as presented in [33].

4.5.3 Attack Budget (Parameters) and Metrics

The proposed crafting algorithm performs optimisation with respect to two types of constraints: feature space (feature budget) and the magnitude of feature change (perturbation budget). We define a feature budget f, based on the $mask$ and w, and calculate it as follows:

$$f = \frac{mask \cdot w}{|w|_1}$$

where $mask \cdot w$ denotes a vector-vector product. If all features of groups G1-3 are used, the value of f is equal to 1, whereas with fewer features it decreases. This value should give an indication of the cost of an attack. Therefore, if an adversary can craft adversarial example with only a few, low-weight features, f is close to 0, hence the cost of the attack is low. Alongside the reduction on the feature space, we restrict our attacks with a maximum perturbation δ_{max}.

To assess the strength of the attack for a given budget f and δ_{max}, we compute the success rate as follows:

$$s_{j\to t} = \frac{1}{N}\sum_{x\in D_j} 1_{F(x+\delta_x)=t} \quad \text{s.t. } \forall \delta_x : \overline{\delta_x} \leq \delta_{max},$$

where F is a given classifier with $F: D \to Y$ (Y being a set of labels), t is the desired label ($t \in Y$) and D_j is the data distribution of instances with true and predicted label j ($D_j \subset D$ with $F(x) = j$ for all $x \in D_j$ and $|D_j| = N$). $1_{\text{condition}}$ is an indicator function, which is 1 if the condition is true, 0 otherwise. The perturbation δ_x, which aims to turn x into an adversarial example is restricted with the $mask$. The perturbation budget (δ_{max}) restricts the average perturbation per feature $\overline{\delta_x}$ of each instance x. As we only consider attacks that aim to be stealthy (target label t is benign), we are going to denote $s_{j\to t}$ as s_j. The overall success rate s would then be the average over all labels j s_j. The success rate only makes sense for a sufficiently large number of test samples N.

Goodfellow *et al.* [15] interpreted adversarial examples as 'blind spots' due to incomplete training data. Using this metaphor, the success rate can intuitively be seen as the relative number of detected blind-spots within a sphere with radius δ_{max} in a hyperspace defined by $mask$. The success rate is dependent on the technique used for crafting adversarial examples and the set of input samples. Therefore, we restrict our evaluation to the empirical success rate, which is the overall success rate for a given crafting algorithm that is tested on a set of samples.

To give an intuition about the vulnerability of the deep learning model against a given attack, we introduce a vulnerability score:

$$vs = \frac{2 \cdot (1/f) \cdot s}{(1/f) + s}$$

which is the harmonic mean between the success rate s and the inverse feature budget f. The vs metric takes values in the range $[0,2]$. The closer vs is to 0, the less vulnerable the model is under test.

This metric is intended to reflect the trade-off between the empirical success rate s and the hyper-space defined by the attack budget.

4.6 Experimental Analysis

In this section, we introduce the experimental setup. We explain and justify the choice of datasets and how they are used for both anomaly detection (binary) and multi-label classification. Furthermore, we present our approach to pre-processing and feature grouping on the datasets. Subsequently, the models are evaluated (their accuracy) and then attacks on those models, to compare their vulnerability (robustness) against the adversarial examples, are performed.

4.6.1 Datasets and Pre-processing

In our experiments, we use two publicly available IDS datasets: NSL-KDD [35], a relatively old but still widely-used benchmark dataset; and CICIDS2017 [36], which is a more recent network flow-based dataset.

The NSL-KDD dataset includes information that has been extracted from network traffic (e.g. TCP connections), as well as high-level features (e.g. the number of failed logins). The data records are attributed with 41 features (intrinsic, content, time-based and host-based). The dataset contains 49 different labels that are grouped into five categories: Normal, Denial of Service (DoS), Probing, User to Root (U2R) and Remote to Local (R2L) attacks. The NSL-KDD dataset is divided into training and testing sets. The NSL-KDD dataset in imbalanced in various ways, e.g. the testing set has more labels than the training set. Consequently, in a similar manner to Rezy *et al.* [29], we create a more balanced dataset for our experiments; this approach is described in our publication [37]. This balanced dataset is referred to as NSL-KDD* hereafter.

The second dataset (CICIDS2017) [36] contains five days of network traffic that were recorded in an emulated environment with 25 hosts. The data is provided as network packets and bidirectional network flows. In total, this set contains about 2.8 million records, which are described with 80 features and grouped into 15 label categories (14 attacks plus benign). The dataset is quite imbalanced -- some attacks are very sparse, e.g. the Heartbleed attack appears only eleven times in the whole dataset, whereas almost 84% of labels are normal samples. In our experiments, we used a subset of the CICIDS2017 dataset (due to computational constraints). We used an 80/20 random split to create training and testing sets. For multi-label classification, we utilized an oversampling technique for the training phase, by adding redundant data to low-frequency labels. We also removed random samples from the benign label (under-sampling). For the anomaly-based detection, we kept the original distribution of the CICIDS2017 dataset, as the benign and total attack samples are sufficiently balanced.

Like Gao *et al*. [30], we have applied three pre-processing methods – these are necessary for the datasets to be used as input to machine learning models. First, we have performed *one-hot-encoding* to map categorical features to numerical values. For instance, the 'Protocol' feature can be of three distinct categories, namely 'TCP', 'UDP' or 'ICMP'. In one-hot-encoding, this feature is represented by a $3x1$ vector, in which the value 'TCP' protocol translates to [1, 0, 0], 'UDP' to [0, 1, 0], etc. This procedure has been applied to the NSL-KDD dataset, encoding 41 categorical features as 122 numerical features. This step was not necessary for the CICIDS2017 dataset, as it does not include categorical features. Second, since the range of the features in the IDS datasets may vary significantly (e.g. some packet-size related features range from 0 to 10^8, others from 0 to 10^3), we apply the following log-transformation to all features: $x_i : x_i = log(1 + x_i)$. Consequently, a value $x_i \in [0, 10^8]$ is transformed to $x_i \in [0,18]$. This approach was applied by Hawkins *et al.* [31]. Finally, we normalize the data

using a *min-max transform*, such that the range of all the features is set to $[0,1]$. Without the log-transformation that is performed in the previous step, outliers would cause most of the features to be set to zero.

4.6.2 Feature Groups

Our approach to creating adversarial examples requires features to be placed into groups, as discussed in Section 3.2. For the CICIDS2017 dataset, we assigned all the flows in a backward direction to G0, along with features that relate to both backward and forward directions (total flow bytes/s). Groups 1, 2 and 3 are features in the forward direction. The independent features are placed in G1, those that are used to derive other features are placed in G2, and derived features in G3. Additionally, 'hard to access' features and those with physical limitations (IAT features) are placed into G3. Meanwhile, for the NSL-KDD dataset, we put all the categorical and binary features (flags) into G0. Content-based features that are based on flows in the forward direction are in G1, alongside (independent) counters. Group 2 contains counters that are used to derive other features. Finally, frequency-based features (e.g. error rates) are placed into G3. We consider 30 out of 122 (25%) features for the NSL-KDD set and 32 out of 78 (41%) for the CICIDS2017 as accessible features, i.e. features that could be manipulated by an adversary. Therefore, the maximum number of features, wherein $f = 1$, can only be achieved by using all 30 and 32 accessible features for NSL-KDD and CICIDS2017 datasets, respectively.

4.6.3 NIDS Model Evaluation

We start our experiments by establishing the baseline performance of the considered algorithms for network intrusion detection. We evaluate both variants of NIDS models: attack detection (multi-label classification models) and anomaly detection (binary classification models).

As mentioned earlier, two classifiers have been trained and tested: a Deep Neural Network (DNN) [29] and a Deep Belief Network DBN [30]. The architecture and training parameters were chosen to be like the proposed models, with slight modifications to optimize the performance. Table 6 shows the performance of the models in terms of overall Accuracy, Recall, Precision and F1-score. The performance of both models is acceptable and aligned with the original papers. It can also be noted that, as expected, both models achieve significantly better accuracy on the NSL-KDD* dataset than NSL-KDD.

Table 6: The results of the attack classification models

Model	Dataset	Accuracy	Precision	Recall	F1-Score
DNN	CICIDS2017	0.979	0.933	0.983	0.957
	NSL-KDD	0.742	0.914	0.603	0.777
	NSL-KDD*	0.973	0.982	0.971	0.976
DBN	CICIDS2017	0.991	0.969	0.996	0.982
	NSL-KDD	0.762	0.929	0.630	0.751
	NSL-KDD*	0.938	0.983	0.908	0.944

For anomaly detection, the DNN model [29] was used again. However, in this case, it was trained for binary classification, i.e. there are only two output classes: benign or attack. The DNN was trained on the NSL-KDD, NSL-KDD* and original subset of the CICIDS2017 (without oversampling) datasets. Table 7 compares the semi-supervised DNN anomaly detection with the unsupervised Autoencoder (AE) model, which was trained only on the set of benign samples from NSL-KDD and CICIDS2017. The AE threshold was set to 80%. The binary DNN achieves slightly better results than the multi-class DNN for

all the datasets. The improvement is especially visible for the original split of the NSL-KDD dataset, which demonstrates that supervised classification is very sensitive to the distribution of labels in the training set. The AE model achieves surprisingly good results for the NSL-KDD data, with precision and recall above 0.88; however, its accuracy on the CICIDS2017 data is drastically lower. The recall of attack detection is just slightly above 50% and low precision indicates a high number of false positives. The reconstruction error of the anomalous instances seems to be insufficient to efficiently differentiate attacks from benign samples. In our evaluation of the adversarial example attacks, we focus only on sufficiently accurate models; hence, we will not consider the AE trained on the CICIDS2017 dataset.

Table 7: The results of the anomaly detection models

Model	Dataset	Accuracy	Precision	Recall	F1-Score
DNN	CICIDS2017	0.990	0.965	0.996	0.980
	NSL-KDD	0.794	0.921	0.698	0.798
	NSL-KDD*	0.975	0.984	0.971	0.978
AE	NSL-KDD	0.870	0.890	0.880	0.885
	CICIDS2017	0.628	0.273	0.548	0.365

The goal with these models is to reproduce a representation of the established deep learning algorithms and test their robustness against adversarial samples. The accuracy and F1-scores in our evaluation demonstrate that the classifiers are reliable. In further experiments, we will use the strongest variants of the classifiers – for the DNN and DBN models, we choose those trained on the CICIDS2017 and NSL-KDD* datasets (as they yield higher accuracy than the ones trained on the original NSL-KDD dataset).

4.6.4 Attack Evaluation

In this section, we evaluate the proposed algorithm for crafting adversarial examples. In this study, our threat model assumes a white-box attack, i.e. the architecture and parameters of the NIDS model are known. The attacker's goal is to launch a stealthy attack using adversarial examples. The idea is to modify the attack sample in such way that a NIDS classifies it as benign. For the multi-label classifiers, we chose the top two attacks (labels with the highest accuracy), as indicated in. The best accuracy of the DNN and DNN models was reported for DoS (99.6% and 99.6%, respectively) and Probe (99.4% and 99.1%) attacks from the NSL-KDD dataset, and DDoS (99.1% and 99.6%) and PortScan (99.8% and 99.0%) from the CICIDS2017 dataset. These four labels are considered for crafting adversarial examples. For anomaly detection, we selected a random subset of the NSL-KDD attack samples. Some assumptions were made in the evaluation of the proposed adversarial example attacks:

- Adversarial examples are only crafted for inputs that are correctly classified with at least 80% confidence.
- The parameter c, which is the trade-off between minimizing the perturbation (distance) and maximizing the loss function, is set using a binary search (see [33]).
- We always list the worst-case attack, thus if not stated explicitly, the parameters are always set to maximise the success rate.
- The perturbation constraint δ_{max} should be understood as the average distance per feature; therefore, the actual perturbation δ must be smaller than $\delta_{max} \cdot |mask|_1$.
- The list of *considered features*, formally defined by $mask$, specifies which features can be modified by the adversarial crafting algorithm. The $mask$ is created by randomly adding

features from groups in the order of their weights (starting with G1 up to G3), until the budget is reached.

- If all features of G1 to G3 are considered then $f = 1$, whereas $f \approx 0$ indicates that only a few features from G1 were selected. For all the classifiers, we used the same $mask$ to get comparable results.

In the following, we experimentally evaluate the strength of the adversarial example attacks for the considered deep learning algorithms under a given budget, expressed in terms of feature space constraints (f) and perturbation magnitude (δ_{max}).

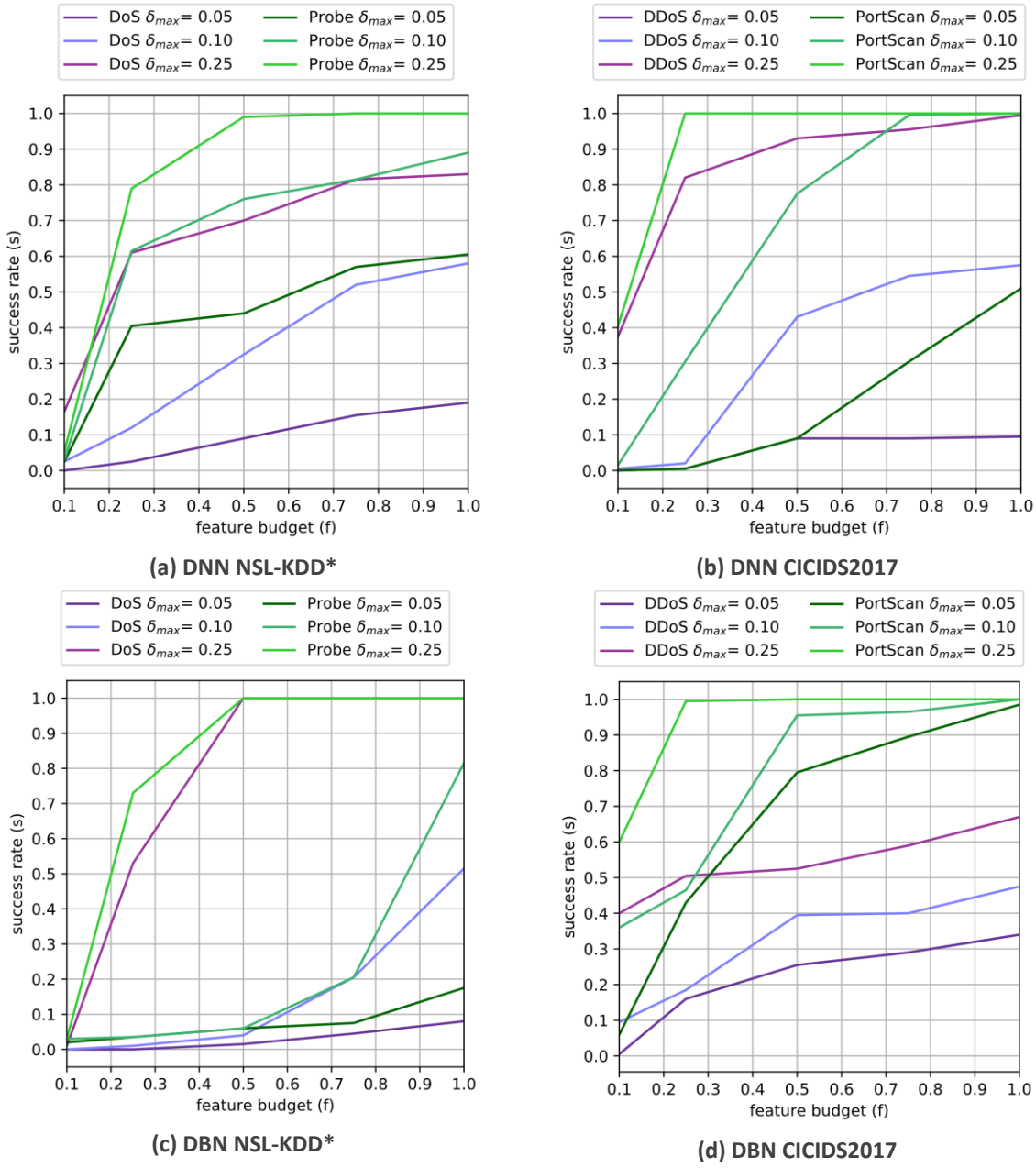


Figure 9: Evaluation of the attack parameters for multi-class models

4.6.5 Attacks on Multi-Class Models

We measure the success rate ($s_{j\to t}$) of the adversarial examples that are crafted by the proposed algorithm, using different combinations of attack parameters. Figure 9 depicts the highest achieved success rate for the DNN and DBN models for $\delta_{max} = \{0.05, 0.1, 0.25\}$ against varying $f = \{0.1, 0.25, 0.5, 0.75, 1\}$. The overall success rate is still quite high, given the fact that we consider only 41% of the features in the CICIDS2017 dataset as accessible and 25% of the NSL-KDD dataset. The results indicate that it is more difficult to craft adversarial examples for DoS attacks. Surprisingly, the DBN model seems to be less vulnerable for low budget attacks than the DNN model for the NSL-KDD* dataset, even though the DNN model outperforms the former in detection. As expected, the success

rate increases with higher values of f – the attacks are more effective if an adversary can manipulate more features.

The plots reveal some intuition about a trade-off between the attack parameters. We can analyse for which δ_{max} an attacker can reach the optimum success rate with a small feature budget. For instance, the most successful attacks can be noted for PortScan against both the DNN and DBN models with $\delta_{max} = 0.25$. Furthermore, we can observe that the success rate of the attack is more sensitive to changes of δ_{max} rather than f. Sometimes we can observe that with enough δ_{max} adding more features does not improve the success rate (e.g. for Probe and PortScan, the optimal feature budget can be found below 0.5). Analogically, for DoS and DDoS attacks, reasonable success rates can only be achieved for the highest δ_{max} value. For most attacks, the optimal feature budget can be found below 0.5; afterwards there is only a gradual, small increase in success rate.

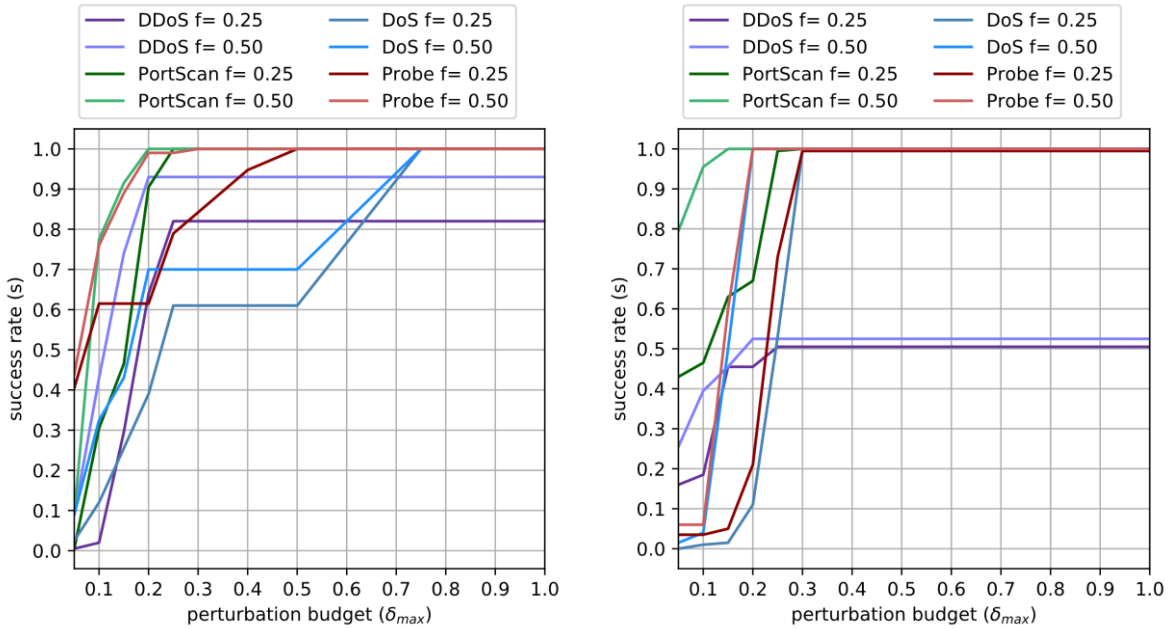
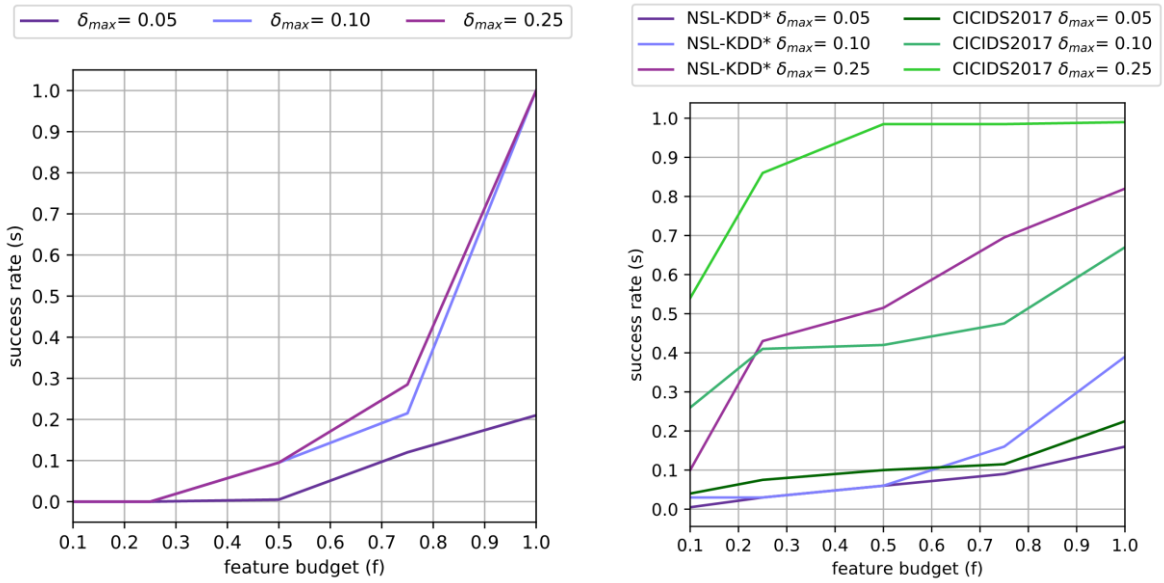


Figure 10: The success rate of attacks against the DNN and DBN models for different values of δ_{max} parameter

We have investigated different values of perturbation budget δ_{max} for $f = 0.25$ and $f = 0.5$. Figure 10 visualizes the influence of δ_{max} on the success rate. We observe that for most graphs there exists a certain threshold after which the success rate stays constant. For the DBN (Figure 10b) model, the optimal value of perturbation budget is reached around $\delta_{max} = 0.3$, whereas for the DNN it is on average slightly higher $\delta_{max} = 0.5$. Overall, this threshold appears to be slightly higher for the NSL-KDD dataset. This is because the relative amount of features we consider is larger for the CICIDS2017 dataset. Therefore, a smaller perturbation-per-feature is necessary to reach the adversarial goal. An average perturbation above 0.3 seems rather extensive; however, we managed to achieve considerable success rates for δ_{max} below this value with $f = 0.5$. This budget for the NSL-KDD* dataset, for example, means that we only perturb around 10% of the features by ≈ 0.1 and still find adversarial examples for many instances.

4.6.6 Attacks on Anomaly-based (Binary) Models

We performed a similar parameter analysis for adversarial example attacks against binary models. We recorded the highest achieved success rate for given perturbation restrictions. The results are depicted in Figure 11.



(a) Autoencoder **(b) Binary DNN**

Figure 11: Evaluation of the attack parameters for binary models

The Autoencoder model treats anomalies as outliers and detects them by measuring the reconstruction error. Figure 11 suggests a rather high budget is necessary to achieve the adversarial goal. Surprisingly, the feature budget (f) seems to have more influence on the success rate than perturbation budget (δ_{max}). Even though the Autoencoder achieves lower accuracy than the other (supervised) models, it has a significant advantage of being more robust to adversarial examples. In Figure 11 the success rates of adversarial example attacks against the binary DNN model are presented. We can clearly observe that attacks using the NSL-KDD* dataset show better performance than CICIDS2017. Furthermore, for the CICIDS2017 dataset it is possible to craft strong attacks even with a small feature budget and relatively small δ_{max}. This might indicate that the distance between benign and anomalous samples is rather low for that dataset, making it challenging for outlier-based anomaly detection. Therefore, the Autoencoder model performed so poorly on the CICIDS2017 dataset (as shown in Table 7).

4.6.7 Vulnerability Analysis

Figure 12 shows a comparison of the algorithms in terms of their vulnerability to adversarial examples. The vulnerability score vs (see Section 4.5.3) is listed per attack against each model. The experiments were performed for three different values of δ_{max}.

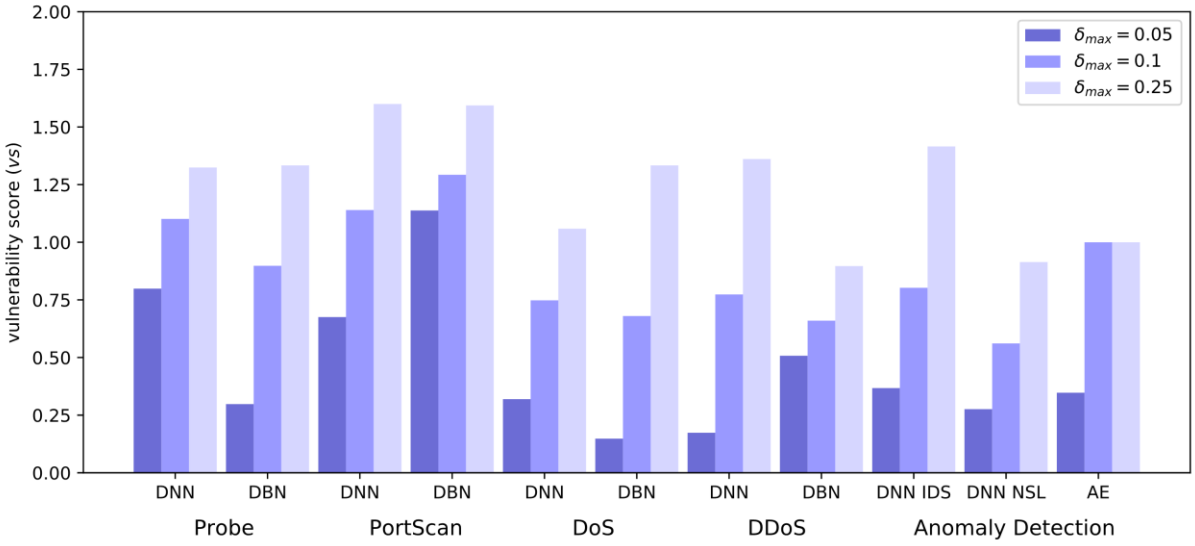


Figure 12: The vulnerability score of all the models for both datasets

We can observe that vs confirms our findings from the previous experiments. The classification models are relatively robust against the DoS attack of the NSL-KDD dataset, but rather vulnerable to the Probe and PortScan attacks. The low vs for the DoS attack may be explained by the fact that the most discriminative features for this type of attack are the inter-arrival times and frequency-based features. However, these features are assigned to G3 in the proposed grouping, thus considered hard to access and modify by an attacker. Considering features with high weights to craft successful adversarial examples targeted towards DoS attacks would increase the overall feature budget f and, therefore, decrease the vulnerability score. As previously shown, the AE model is robust against low-budget adversarial attacks. This is reflected by $vs \leq 1$ for all values of δ_{max}.

4.6.8 Feature Perturbation Analysis

To show the impact of the feature weights in the proposed algorithm, we performed a feature perturbation analysis. For this experiment, we used the DNN classifier trained on the CICIDS2017 dataset and crafted adversarial examples for the DDoS label. We compare the proposed crafting algorithm with the original C&W [18] and FGSM [38] attacks. The latter attack performs iterative FGSM [15], which maximises the loss using gradient sign. This algorithm terminates when either the desired label is reached, or the limit of maximum iterations is exceeded; hence, it does not perform any distance (perturbation) minimisation. Therefore, we neglect the restriction on perturbation budget (maximum distance constraint) in the C&W and our method, since this restriction is not directly relevant for the analysis. As considered features, four features of each group have been chosen arbitrarily. The resulting feature budget sums up to $f = 0.31$. We measure the average perturbation per each feature for the successfully created adversarial examples. The results are depicted in Figure 13.

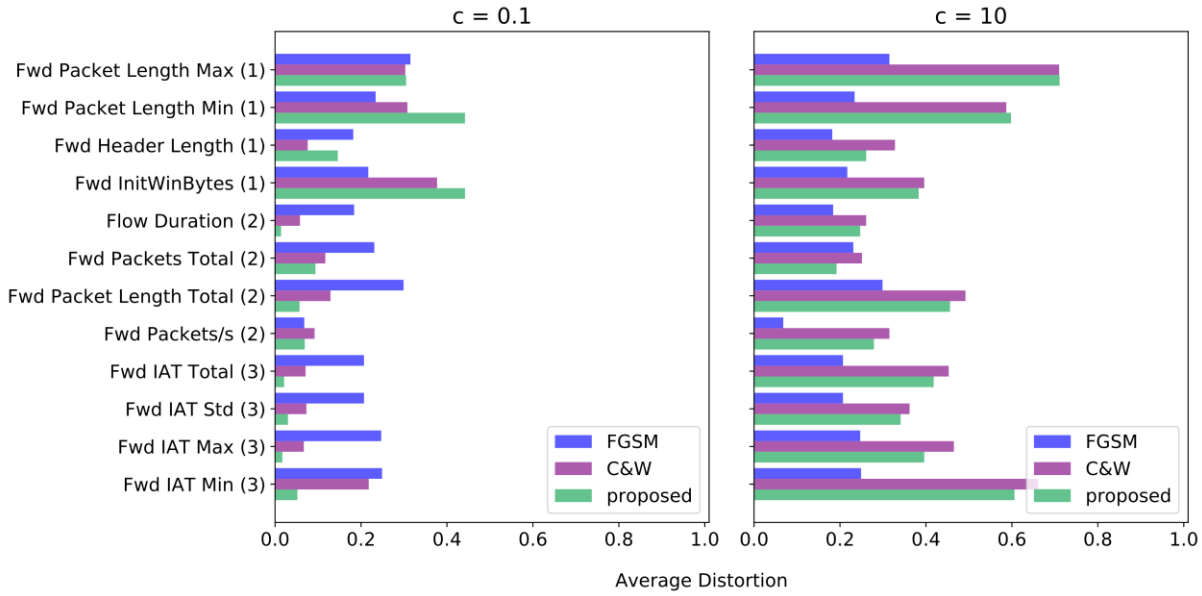


Figure 13: The average distortion per feature for the DDoS attack label -- feature group numbers are shown in parentheses

As expected, the average perturbation for FGSM and C&W does not depend on the feature group but feature importance with respect to the classifier. Furthermore, for C&W and our proposed extension, we compare the average perturbation per feature using two different values of the hyperparameter c, which controls the impact between distance and target function minimisation. A low value of $c = 0.1$, assigns less importance to target function minimisation and enforces a small distance over desired attack, hence the weights of the extended C&W algorithm strongly influence the results. As can be observed in Figure 13 (left), our algorithm favours features from G1 over the other groups. This effect is less visible with higher $c = 10$, in which adversarial examples are optimised mostly with respect to target function. The success rate of the FGSM in Figure 13 was 73.3% (left and right). With emphasis on the distance minimization (left, $c = 0.1$) our proposed algorithm reached a success rate of 50.8%, the original C&W 55.6%, even though the proposed algorithm applied only a perturbation close to 0 to the G3 features. By increasing c to 10, we achieved 87% for the proposed method and 87.7% for the C&W.

For an adversary that intends to attack a NIDS through adversarial examples, it is vital to account for not only the feature space constraints (e.g. only a subset of features might be accessible) but also perturbation restrictions that are specific for some features. We demonstrate that using our weighted C&W approach, we can control the amount of perturbation per group of features. Our results show that the proposed method kept the perturbation of IAT features (G3) as low as possible, since these features are important for maintaining the goal of the DDoS attacks. In our experiments, we assigned weight values to $v = \{1,2,3\}$ for Group 1, 2 and 3, respectively. However, the impact of weighted perturbation could be amplified by using different weight ratios.

5 Conclusion

In this deliverable, we have explored two forms of adversarial machine learning that target models that are used for cybersecurity – one to detect malware-generated DNS names and a second set of models that perform network intrusion detection. For the former study, we have demonstrated that it is possible to generate well-formed DNS names that are compliant with relevant standards (i.e. IETF

RFCs) that have the characteristics of a malware-generated domain name and cannot be detected by our model. The model in question was developed as part of the SOCCRATES project – the DGA Detective [1] – and was trained on a very large dataset that is available to the project from the Shadowserver Foundation. Despite this large and representative dataset, the genetic algorithm that we used was still able to successfully generate names that may not be detected. The purpose of this study is, on the one hand, to evaluate the robustness of the model generated to this form of attack and, on the other, highlight the need to train models in an adversarial setting. More specifically, the model that was trained using the Shadowserver Foundation data can be retrained (or updated) using the dataset that was generated by the genetic algorithm in our study, thus rendering it more robust to this form of attack. We advocate that this form of adversarial training will become increasingly important (and required) as threat actors start to use this form of attack to subvert machine learning models, e.g. to evade detection.

The second study that is described in this deliverable focuses on the emerging threat of adversarial examples. This threat has been explored most extensively in the image classification domain, in which there are potential safety and security concerns if, for example, these attacks are targeted towards autonomous vehicles that use image recognition technology and physical security systems that leverage similar technology [28], [39]. Here we explored the nature of this threat for network-based intrusion detection systems that use a variety of models to perform binary classification (i.e. intrusion detection) and multi-class classification (determining the specific attack type). Whilst the results of our study require the adversary to have a great deal of information about the model that is being targeted – whitebox knowledge – it shows that it is possible to generate adversarial examples that can cause misclassification. We show that certain models are vulnerable to this form of attack and it is easier or otherwise for an adversary to have certain attack types misclassified. However, the overall conclusion is that there is the potential for an adversary to use this type of attack in the future and further study is required.

For both studies, we had to explore the nature of the constraints that were placed on the adversarial machine learning attacks. For example, for the DNS names, we stipulated they must be well-formed names (according to the pertinent RFCs) and have the appearance of those generated by malware (i.e. we could not simply use 'normal'-looking names to evade detection). Meanwhile, for the adversarial examples, we extended a threat model that constrained the adversary to only generate attacks that could be realized in practice (i.e. we could not manipulate flow features in the reverse direction) and resulted in the attack's intended impact (e.g. a DoS attack that generates "adversarial example" features still results in resource starvation at the target). Moreover, we specified a perturbation budget (δ_{max}) that constrained the "distance" the algorithms could change the features from their original values. In both cases, these constraints can be thought of as relating to *function* (well-formed DNS names and protocol compliance) and *perceptibility* (i.e. whether the test data is noticeably changed). For adversarial examples in the image classification domain, the concern relates only to perceptibility – there are no comparable functional requirements to speak of. Moving forward, as the research community explores this challenge for other types of system and data, domain-specific constraints will have to be considered. A common framework would be beneficial that seeks to achieve consensus on these constraints for different systems – such a framework would support the development of tools and testing approaches to improve the robustness of machine learning algorithms to these forms of attack.

The study on adversarial examples for NIDS generates several open questions, specifically around their practical use by attackers. Arguably, in this study – and others like it – we have highlighted the theoretical feasibility of such attacks but do not examine how practicable they are and whether adversaries would have the motivation and capabilities to execute them. To appreciate the capabilities that are required for an adversary to execute an attack, more insights are required about how they could be realized. We propose that future research is needed on this issue, which considers the nature of the system under attack. For example, it is perhaps significantly more challenging to conduct the attacks that are described herein, which target a NIDS, than a similar attack that targets a host-based IDS that uses machine learning (the rationale being it is easier to gain access to model inputs and outputs to train a surrogate model for a host-based IDS than a NIDS).

Finally, a goal of this study was to highlight the importance of training machine learning models – especially those applied to critical tasks like cybersecurity – in adversarial settings, such as those explored in this deliverable. For the moment, this is still a relatively immature topic, compared to other aspects of cybersecurity. There is an opportunity to make advances on approaches to hardening machine models, whilst they are not the focus of attackers. Moreover, there is an imperative to develop best practices for the adversarial training of machine learning models and the means to request these practices from suppliers. Therefore, adversary-aware machine learning should be considered necessary, which means that the models must be evaluated in adversarial settings, tested with the latest attacks and kept up-do-date in terms of pro-active defences.

6 References

[1] P. Kijewski, I. Chiscop, F. Falconieri, and F. Soro, "D4.3 Threat Identification and Threat Trend Prediction – Final Prototype," Feb. 2022. [Online]. Available: https://www.soccrates.eu/result_category/deliverables/

[2] E. Piatkowska *et al.*, "D4.1 - AI-based Attack Detection to Detect Advanced Threats," Aug. 2020. [Online]. Available: https://www.soccrates.eu/result_category/deliverables/

[3] B. Dong and X. Wang, "Comparison deep learning method to traditional methods using for network intrusion detection," in *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, 2016, pp. 581–585.

[4] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection," *IEEE Commun. Surv. Tutor.*, pp. 686–728, 2019.

[5] C. Szegedy *et al.*, "Intriguing properties of neural networks," *ArXiv13126199 Cs*, Dec. 2013, Accessed: Jun. 14, 2018. [Online]. Available: http://arxiv.org/abs/1312.6199

[6] X. Wang, J. Li, X. Kuang, Y. Tan, and J. Li, "The security of machine learning in an adversarial setting: A survey," *J. Parallel Distrib. Comput.*, vol. 130, pp. 12–23, 2019, doi: https://doi.org/10.1016/j.jpdc.2019.03.003.

[7] A. AlEroud and G. Karabatis, "Bypassing Detection of URL-Based Phishing Attacks Using Generative Adversarial Deep Neural Networks," in *Proceedings of the Sixth International Workshop on Security and Privacy Analytics*, New York, NY, USA, 2020, pp. 53–60. doi: 10.1145/3375708.3380315.

[8] C. Sitawarin, S. Chakraborty, and D. Wagner, "SAT: Improving Adversarial Training via Curriculum-Based Loss Smoothing," in *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, New York, NY, USA, 2021, pp. 25–36. doi: 10.1145/3474369.3486878.

[9] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[10] S. Yadav, A. K. K. Reddy, A. L. N. Reddy, and S. Ranjan, "Detecting Algorithmically Generated Malicious Domain Names," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, New York, NY, USA, 2010, pp. 48–61. doi: 10.1145/1879141.1879148.

[11] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Detecting Homoglyph Attacks with a Siamese Neural Network," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018, pp. 22–28. doi: 10.1109/SPW.2018.00012.

[12] S. Krishnan, T. Taylor, F. Monrose, and J. McHugh, "Crossing the threshold: Detecting network malfeasance via sequential hypothesis testing," *2013 43rd Annu. IEEEIFIP Int. Conf. Dependable Syst. Netw. DSN*, pp. 1–12, 2013.

[13] P. Mockapetris, *RFC 1035 Domain Names - Implementation and Specification*. Internet Engineering Task Force, 1987. [Online]. Available: http://tools.ietf.org/html/rfc1035

[14] S. Bai, J. Z. Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling." 2018.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *ArXiv14126572 Cs Stat*, Dec. 2014, Accessed: Sep. 20, 2018. [Online]. Available: http://arxiv.org/abs/1412.6572

[16] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," in *arXiv:1511.07528 [cs, stat]*, Mar. 2016, pp. 372–387.

[17] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2574–2582.

[18] N. Carlini and D. Wagner, "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods," *ArXiv170507263 Cs*, May 2017, Accessed: Jun. 14, 2018. [Online]. Available: http://arxiv.org/abs/1705.07263

[19] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial Examples: Attacks and Defenses for Deep Learning," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–20, 2019.

[20] N. Carlini *et al.*, "On Evaluating Adversarial Robustness," *ArXiv190206705 Cs Stat*, 2019.
[21] K. Yang, J. Liu, C. Zhang, and Y. Fang, "Adversarial Examples Against the Deep Learning Based Network Intrusion Detection Systems," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 559–564.
[22] X. Zhang, Y. Zhou, S. Pei, J. Zhuge, and J. Chen, "Adversarial Examples Detection for XSS Attacks Based on Generative Adversarial Networks," *IEEE Access*, 2020.
[23] M. Hashemi, G. Cusack, and E. Keller, "Towards Evaluation of NIDSs in Adversarial Setting," in *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks - Big-DAMA '19*, 2019, pp. 14–21.
[24] T. Weng *et al.*, "Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach," *ArXiv180110578 Cs Stat*, 2018.
[25] A. Hartl, M. Bachl, J. Fabini, and T. Zseby, "Explainability and Adversarial Robustness for RNNs," *ArXiv191209855 Cs Stat*, 2020.
[26] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples," *ArXiv160507277 Cs*, 2016.
[27] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into Transferable Adversarial Examples and Black-box Attacks," *ArXiv161102770 Cs*, 2016.
[28] K. Eykholt *et al.*, "Robust Physical-World Attacks on Deep Learning Models," *ArXiv170708945 Cs*, 2017.
[29] S. Rezvy, M. Petridis, and A. Lasebae, "Intrusion detection and classification with autoencoded deep neural network," *Springer Int. Publ.*, 2018.
[30] N. Gao, L. Gao, Q. Gao, and H. Wang, "An Intrusion Detection Model Based on Deep Belief Networks," in *2014 Second International Conference on Advanced Cloud and Big Data*, 2014, pp. 247–252.
[31] S. Hawkins, H. He, G. Williams, and R. Baxter, "Outlier Detection Using Replicator Neural Networks," in *Data Warehousing and Knowledge Discovery*, 2002, pp. 170–180.
[32] M. Azami, C. Lartizien, and S. Canu, "Converting SVDD Scores into Probability Estimates," *Comput. Intell.*, 2016.
[33] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," *ArXiv160804644 Cs*, Aug. 2016, Accessed: Jun. 14, 2018. [Online]. Available: http://arxiv.org/abs/1608.04644
[34] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ArXiv14126980 Cs*, 2017.
[35] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," in *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.
[36] I. Sharafaldin, A. Habibi, and A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, 2018, pp. 108–116.
[37] M. Teuffenbach, E. Piatkowska, and P. Smith, "Subverting Network Intrusion Detection:\ Crafting Adversarial Examples\ Accounting for Domain-Specific Constraints," 2020.
[38] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale," *ArXiv161101236 Cs Stat*, 2017.
[39] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, Vienna, Austria, 2016, pp. 1528–1540. doi: 10.1145/2976749.2978392.

7 Abbreviations

This glossary serves as inventory of abbreviations used in the document.

Acronym	Description
AI	Artificial Intelligence
ACT	semi-Automated Cyber Threat intelligence
ADG	Attack Defence Graph
AE	Autoencoder
AEF	Argus Event Format
AI	Artificial Intelligence
AIT	AIT Austrian Institute of technology
API	Application Programming Interface
APT	Advance Persistent Threat
ATOS	ATOS Spain
AV	AntiVirus
BPMN	Business Process Model and Notation
C&W	Carlini and Wagner
CC	Command and Control
CERT	Computer Emergency Response Team
CMDB	Configuration Management Database
CSIRT	Computer Security Incident Response Team
CoA	Course of Action
CTI	Cyber Threat Intelligence
DC	DataCentre
DGA	Domain Generation Algorithm
DBN	Deep Belief Network
DNN	Deep Neural Network
DNS	Domain Name System
EDR	Endpoint Detection and Response
ELK	Elasticsearch/Logstash/Kibana
FGSM	Fast Gradient Sign Method
FRS	Foreseeti
FSC	F-secure
GA	Genetic Algorithm
GAN	Generative Adversarial Network
IAT	Inter-Arrival Time
ICT	Information and Communication Technology
IDS	Intrusion Detection System
IMC	Infrastructure Modelling Component
IMT	Institut Mines Télécom – Télécom SudParis
INTF	Interface
IoC	Indicators of Compromise
IP	Internet Protocol
IPS	Intrusion Prevention System
IRM	Incident Response and Management
ITIL	Information Technology Infrastructure Library

KTH	Kungliga Tekniska Högskolan - Royal Institute of Technology
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
M_n	Infrastructure Model (at time *n*)
ML	Machine Learning
MNM	mnemonic
MSSP	Managed Security Service Provider
MTTD	Mean Time To Detection
NOC	Network Operations Centre
NIDS	Network Intrusion Detection System
OT	Operational Technology
OS	Operating System
RFC	Request For Comment
RORI	Return on Response Investment
SDN	Software Defined Network
SHS	Shadowserver
SIEM	Security information and event management
SOAR	Security Orchestration, Automation and Response
SOC	Security Operation Centre
SOCCRATES	SOC & CSIRT Response to Attacks & Threats based on attack defence graph Evaluation Systems
SSL	Secure Sockets Layer
TAP	Test Access Point
TIP	Threat Intelligence platform
TLS	Transport Layer Security
TNO	Nederlandse Organisatie voor toegepast natuurwetenschappelijk onderzoek
TTC	Time To Compromise
UC	Use Case
VLAN	Virtual LAN
VM	Virtual Machine
VTF	Vattenfall