Towards Evaluation of NIDSs in Adversarial Setting

Mohammad J. Hashemi
mohammad.hashemi@colorado.edu
University of Colorado Boulder
Boulder, Colorado

Greg Cusack
gregory.cusack@colorado.edu
University of Colorado Boulder
Boulder, Colorado

Eric Keller
eric.keller@colorado.edu
University of Colorado Boulder
Boulder, Colorado

ABSTRACT

Signature-based Network Intrusion Detection Systems (NIDSs) have traditionally been used to detect malicious traffic, but they are incapable of detecting new threats. As a result, anomaly-based NIDSs, built on neural networks, are beginning to receive attention due to their ability to seek out new attacks. However, it has been shown that neural networks are vulnerable to adversarial example attacks in other domains. But, previously proposed anomaly-based NIDSs have not been evaluated in such adversarial settings. In this paper, we show how to evaluate an anomaly-based NIDS trained on network traffic in the face of adversarial inputs. We show how to craft adversarial inputs in the highly constrained network domain, and we evaluate 3 recently proposed NIDSs in an adversarial setting.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; • **Computing methodologies → Neural networks**.

KEYWORDS

Intrusion Detection Systems; Neural Networks; Anomaly Detection; Adversarial Example

ACM Reference Format:
Mohammad J. Hashemi, Greg Cusack, and Eric Keller. 2019. Towards Evaluation of NIDSs in Adversarial Setting. In *Big-DAMA '19: ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks, December 9, 2019, Orlando, FL, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3359992.3366642

1 INTRODUCTION

Network attacks continue to grow in complexity, scale, and number [24]. The impacts of these attacks range from high monetary costs for exploited businesses and individuals, to more serious issues, such as wide-scale power outages [14]. Due to the growing number of attacks, and their severe, adverse effects, companies are expected to invest billions of dollars by 2021 to find effective tools that detect and eliminate network intrusions [15].

Network intrusion detection systems (NIDSs) are one part in the line of defense against network attacks. Historically, these are based on signatures – whether it be a known sequence of bytes (with deep packet inspection) or known and fixed access patterns. While providing a degree of protection for networks, this approach has a problem in that it relies on the signatures of *known* attacks.

In response, there has been a great deal of research. and even commercial offerings, that leverage machine learning (with deep neural networks) to augment the detection capabilities [1, 7, 17, 18, 26–29]. These anomaly-based NIDSs have been introduced due to their ability to detect zero-day attacks (for which there is no pre-existing signature) by looking for deviations from typical, benign network traffic. To do so, these NIDSs are trained only on benign traffic. Then, during inference time, the NIDS measures how similar the new traffic is to the traffic seen during training time. Each packet or flow seen by the NIDS is given a similarity score and compared to a predefined threshold. If the packet or flow score exceeds the threshold, the traffic is considered malicious.

While moving toward deep neural networks for NIDSs holds great promise, there is an underlying problem that has yet to be addressed, their vulnerability to adversarial examples. Previous work in other domains (e.g., image classification) has shown that neural networks are vulnerable to adversarial example attacks [4, 25], small perturbations of the input that can bypass or purposely alter the classification. In the case of images, this might be changing a few pixels (imperceptible to the human eye) such that the classifier misclassifies a specific person as a different person or hides that person all together. Unfortunately, we don't fully understand the implications in the context of NIDSs because previously proposed, anomaly-based NIDSs have not been evaluated in adversarial settings [1, 7, 17, 18, 26–29]. The other downside of these anomaly-based NIDSs is that they are evaluated on outdated datasets [7].

In order to address these issues, we introduce a technique to evaluate anomaly-based NIDSs in an adversarial setting. We also perform an evaluation of previously proposed NIDSs with this technique on a new dataset that contains 12 different network attacks. To do so, we needed to overcome some challenges not seen in other domains. When generating adversarial examples, we are constrained by two key factors: (i) we must retain the network protocol correctness, and (ii) we must retain the attack's semantics. Therefore, in this paper, we illustrate how to craft adversarial examples for networks by identifying traffic manipulations that can change the network features but remain within the constraints above. In summary, we make the following contributions:

- For the first time, we explain how an adversary can legitimately modify network traffic in order to fool an anomaly-based NIDS and not break underlying network protocols.
- We show how these transformations can be tailored towards a packet-based NIDS, which predicts the malicious traffic in real-time by extracting features from each packet.
- We demonstrate how an adversary can fool a flow-based NIDS that detects malicious traffic based on the high-level

Big-DAMA '19, December 9, 2019, Orlando, FL, USA

ACM ISBN 978-1-4503-6999-2/19/12…$15.00
https://doi.org/10.1145/3359992.3366642

features extracted from the whole flow by considering the legitimate transformations we introduce.
- We evaluate the aforementioned NIDSs on a new network traffic dataset, which contains a wide range of attacks, to show how each of these attacks can be maliciously modified to fool an NIDS.

2 BACKGROUND

2.1 Anomaly-based NIDSs

Anomaly-based NIDSs can be built in many different ways. In general, for each input, they calculate a score, and if that score is higher than a predefined threshold, they consider the input malicious. We categorize these anomaly-based NIDSs into two different groups: packet-based NIDSs and flow-based NIDSs. A packet-based NIDS outputs a score for each packet that it receives. This decision is not necessarily based solely on the current packet; it can also consider the history of packets it has seen earlier. In contrast, flow-based NIDSs make decisions based on features extracted from a whole flow and mark the whole flow as malicious or benign. For the evaluation of packet-based NIDSs we consider Kitsune [18], and for the evaluation of flow-based NIDSs we consider DAGMM [29] and a BiGAN-based anomaly detector [28]. We refer the readers to these papers for more details about each approach. These three NIDSs are among the most cited anomaly detectors which have been published recently. They also leverage very different techniques to detect anomalies. Kitsune builds manual features which are extracted from the sequence of packets and keeps some internal state for each flow. The other two are stateless. DAGMM [29] calculates the energy of each flow in the Gaussian Mixture Model (GMM) framework and uses that energy to detect anomalies, while the BiGAN-based approach [28] uses the reconstruction error of the generator and the output of the discriminator in the GAN framework to detect anomalies. Therefore, by evaluating our attacks against these 3 NIDSs we also show its potential to craft adversarial examples in general.

2.2 DNNs in Adversarial Settings

It has been shown that deep neural networks used for tasks such as image classification, speech recognition, etc. are vulnerable to adversarial example attacks (i.e. evasion attacks) [4, 25]. Adversarial example attacks are carried out during the inference phase. For this attack, the attacker doesn't change any of the model's parameters but modifies its own inputs in a way to make the model predict the inputs as the desired class.

Szegedy et al. in [25] and Biggio et al. in [4] showed the vulnerability of image classifiers to adversarial examples in the white-box setting. The procedure to craft an adversarial example against an image classifier can be formulated as a box-constraint optimization problem as follows:

$$argmin_{\delta} ||\delta||_p \text{ s.t. } (x+\delta) \in [0,1]^m \text{ and } F(x+\delta) = y_{target}$$

in which x is the legitimate input, δ is the perturbation added to a legitimate input to make it adversarial. $F(.)$ is the classifier which maps an image to a label. y_{target} is the attacker's desired class and m is the number of pixels in the image. In order to solve this minimization problem, different methods have been designed; they can be found at the following references: [3, 5, 9, 19, 21, 25].

It has also been shown that adversarial examples can be crafted against image classifiers in the black-box setting [4, 16, 20]. Recently, many defenses have been proposed to make these models robust against adversarial examples. However, since the release of the defenses, other researchers have introduced new attacks to bypass them [2, 13]. It has also been shown that adversarial examples exist in the areas beyond digital images. Sharif et al. in [23] showed how an adversary can print a sticker and add it to glasses to fool face recognition systems in the physical world. Later, Eykholt et al. in [8] showed that an adversary can place a few stickers on a stop sign to fool a classifier potentially deployed on a moving vehicle, *e.g.,* causing the classifier to predict a stop sign as a speed limit sign. Carlini et al. in [6] illustrated how an adversary can craft adversarial examples against speech-to-text systems. Grosse et al. in [10] also showed it is possible to craft adversarial examples against malware detectors. Note that crafting adversarial examples in each of these domains introduces its own problems that need to be tackled to successfully fool the target model. For example, as is shown in the work done by Sharif et al. [23], attacks designed for digital images could not fool the target face recognition system in the physical world. This is due to the fact that printers are unable to print every pixel value of a digital image. As a result, the authors added a constraint to their adversarial example generation, which dictated that pixels could only be changed to a color that a printer could print. Similarly, crafting adversarial examples against NIDSs introduces its own challenges which we will discuss in the next section.

3 NIDS IN ADVERSARIAL SETTING

3.1 Threat Model

Before outlining our approach, we define the threat model we consider in evaluating anomaly-based NIDSs. Figure 1 provides an overview of our threat model and system overview. In order to have a complete evaluation, we consider a white-box setting. That is to say, we consider that the attacker has a copy of the NIDS deployed on the victim network and knows all of its parameters. The NIDS deployed on the victim network receives a copy of all the packets that travel through the network entrances (⑤). We also consider that attacker's resources are limited to what she already used to create the original attack. In other words, in order to generate the adversarial version of a network attack, we assume the attacker does not want to use more machines or more IP addresses. The attacker also is considered to be outside of the victim's network (①).

3.2 Challenges in Crafting Adversarial Examples for NIDS

Crafting adversarial examples against NIDSs that are trained on network traffic introduces its own complications and constraints. Thus, the crafting procedure needs to be tailored for NIDSs. Here, we mention some of the differences that exist between images and network traffic that prevent an adversary from fooling the NIDSs with the same procedure used against image classifiers. First of all, pixels in an image can be modified freely. This is not the case for

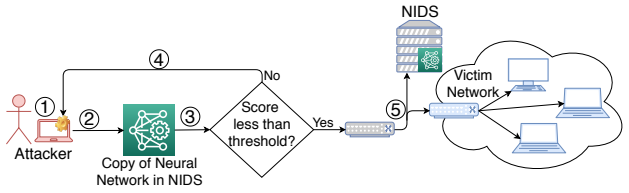

Figure 1: System overview and threat model considered when evaluating and designing anomaly-based intrusion detection systems. ①: The attacker sits outside the victim network and generates adversarial examples. ②: Adversarial examples are sent to the local copy of the NIDS for evaluation. ③: A classification score is produced by the NIDS based on the input. If the output score is greater than the threshold, the attacker applies some modifications, ④, to improve the adversarial example. This loop back process is carried out a maximum of N times. If the score in ③ is less than the threshold, the packet is mirrored to the NIDS and sent to the victim network ⑤.

a sequence of network packets. For example, if features that are fed into an NIDS are packet headers, changing some of the headers could cause the communication between the attacker and the victim to breakdown. To this extent, during the crafting procedure, attackers should ensure that the communication channel does not timeout or breakdown. Second, pixels in an image can be modified independently of each other. This is not true for typical features fed into an NIDS. In many cases these features are dependent on each other, and there is no guarantee that a valid network flow exists that matches the features generated by the crafting procedure. For example, a flow's average interarrival time between packets is directly tied to the flow's duration and number of packets through the following relationship: $Flow\_IAT_{avg} = \frac{Duration_{Flow}}{Pkt\_count_{Flow}-1}$

As a result, we cannot arbitrarily change these features independently. We must ensure the inherent properties of flows are not violated. In addition, all adversarial image pixels can be modified to fool an image classifier, but this is not the case for NIDSs. Many of the features that are fed into them are extracted from the packets generated by the victim. These are packets that the attacker doesn't have control over. The differences between adversarial image generation and adversarial network traffic generation along with the security concerns that fooling an NIDS raise, demonstrate the need to explore how an NIDS can be evaluated in an adversarial setting.

3.3 Legitimate Packet Transformations

If we are able to manipulate the malicious packets of an attack to have specific features that mimic benign traffic, we will be able to bypass NIDSs.

We declare an attack a success if the manipulated attack packets meet the following three requirements.

(1) The packets must carry out their original malicious intent effectively (e.g. a port scan, after transformation, should scan the victim's ports).
(2) Packet transformations must not break the underlying protocols the attack relies on (e.g. a TCP-based attack cannot violate TCP).
(3) The attack must not be flagged as an intrusion by the anomaly-based NIDSs. We will evaluate this requirement for existing systems in Section 5.

From these requirements and from studying the features used in existing anomaly-based NIDSs, we identify three, general, packet manipulation techniques that can be used for crafting adversarial versions of network attacks.

The manipulations are as follows:

- **Split**: The attacker can increase the number of packets sent by splitting the original payload of each packet across multiple packets. For TCP, as long as sequence numbers, acknowledgement numbers, and IP IDs are updated properly, the attack remains effective as no information is lost and the packets are reassembled at the victim host.
- **Delay**: The attacker may adjust the time between outgoing packets by either increasing or decreasing the time elapsed between subsequent packets. Since the packets themselves are not modified, the attack will not only maintain its effectiveness (so long as there is not a connection timeout), but it will also adhere to the underlying network protocols.
- **Inject**: The attacker also has the ability to construct fake packets with arbitrary lengths, transmission times, and flag combinations. She can send the decoy packets among the real attack packets as long as she can ensure that these fake packets are ignored by the victim but processed by the NIDS. By doing so, an NIDS takes into account packets that both reach and don't reach the victim into its decision on whether or not the current flow is malicious. The attacker can rely on the fundamentals of TCP, UDP, and IP protocols to guarantee these decoy packets are processed by the NIDS but not by the victim host. For example, the attacker can inject a TCP packet with a sequence number smaller than the ACK number acknowledged by the victim. Furthermore, by setting the TTL field of the IP header such that the TTL is greater than zero when processed by the NIDS but decrements to zero prior to reaching the victim, the attacker ensures the packet is dropped after reaching the NIDS but before the victim.

Therefore, in order to fool an NIDS which is trained on network traffic packets, the adversary should modify the malicious traffic with a set of legitimate transformations as described above. In the next section, we describe how we can use these transformations to attack several NIDSs.

4 CRAFTING ADVERSARIAL EXAMPLES

In this section, we first explain how to tailor the legitimate transformations introduced in the previous section towards packet-based NIDSs, and then move on to flow-based NIDSs.

4.1 Adversarial Examples for Packet-based NIDSs

Algorithm 1 shows how we tailored legitimate transformations, introduced in the previous section, towards Kitsune. In a nutshell, Kitsune keeps some internal states for each flow and each packet

moves through the network, updates the corresponding state. Then it calculates a score, based on features extracted from the internal state to decide whether the current packet is from a malicious traffic or not. In order to fool Kitsune, Each malicious packet that is sent from the attacker, is fed through the local neural network copy and the output score is registered. If the score of that packet is close to the threshold found during training time, we see if waiting a few moments can help reduce its score. More specifically, we implement the TryDelay procedure, which performs a binary search in the range between 0 and 15 seconds to see if adding a delay can bring the score of the current packet to less than $0.9 \times threshold$. In the case that the score is greater than the threshold, we also try splitting the packet.

The TrySplit procedure tries to convert a large packet into multiple smaller packets such that the score of all of them becomes smaller than the threshold. Since we don't know what the right cut-offs are to split the original packet, we search for the correct cut-off by trying different values. More specifically, we split the payload of packet with L bytes into two packets with r and $L-r$ bytes of payload, where r is chosen randomly. Since this cutoff might not be the right one, we need to backup the state of the local NIDS related to the current flow and restore it in case the split failed. When this happens, we try a different r. We need to do checkpoint the NIDS's state to make sure that the state of local copy remains the same as the remote NIDS. If the first portion (r bytes) of payload could fool the NIDS, we would do the same thing for the second part (the remaining $L-r$ bytes) recursively until the whole packet's payload would be sent and none of them would be detected. Finally, if delaying or splitting the original packet could help to fool the local copy, the attacker will make the appropriate change(s) and send the packet(s) to the victim. Otherwise, the original packet would be sent.

If the malicious packet is sent from the victim and its score is larger than the threshold, the only thing the attacker can do is to change the state of the NIDS such that the victim's packet do not pass the threshold. In this case, we see if injecting a fake packet from the attacker before the victim's packet can fool the NIDS for both packets such that the score of both of them becomes less than threshold. More specifically, in the TryInject procedure, we send a packet from the attacker with different payload sizes. If that packet's score is less than the threshold, we send the victim's packet after that. If the score of both packets is less than the threshold, we inject that packet, otherwise we restore the state of the local NIDS to the state before sending the fake packet. We repeat this for another fake packet with different length. Also, since the TryInject procedure is a slow process, we run it occasionally. We keep track of the times that TryInject succeeds and fails for each attack. Then, for each new packet from victim, we run the TryInject procedure with the probability of $\delta = \frac{\#successes}{(\#successes+\#failures)}$. After each success, we reset δ to one. In practice, this means that, given a network attack, if TryInject does not work for a while, we run it less frequently. If suddenly it succeeds for a packet, we again try it on consecutive victim's packets more frequently.

Algorithm 1 Crating adversarial examples for Kitsune

```
1:  procedure CraftAdvEx(x)                          ▷ x is a malicious packet
2:      if x is sent from the attacker then
3:          if score_x > 0.9 × threshold then
4:              TryDelay(x)
5:              if score_x > threshold then
6:                  TrySplit(x)
7:              end if
8:              Send the split packets with appropriate delay if successful.
9:          end if
10:     else                                         ▷ x is sent from victim
11:         TryInject(x)
12:         Send the fake packet before victim's packet if successful.
13:     end if
14: end procedure
```

4.2 Adversarial Examples for flow-based NIDSs

In order to evaluate flow-based NIDSs in an adversarial setting, we group the features fed into them into 4 different groups. As we mentioned earlier, manipulating the features fed to an NIDS in an adversarial manner is different from changing the pixels of an image. Here we consider two of the main differences. One difference is that some of the flow-based features can't be changed because the attacker doesn't have control over them since they are extracted from the victim's traffic. Also, some features depend on other features. For example, the mean of packet payloads in the forward direction can be calculated based on two other features, total length of payloads in the forward direction and the total number of forward packets. There is another type of feature in which their value depends on the actual packets of the flow and cannot be calculated by the value of other features (e.g., std of packet payloads in forward direction). As a result, we group flow features into the following four groups.

(1) Features that should not be changed because they are extracted from backward flowing packets (victim packets).
(2) Features that can be changed independently of each other by using the legitimate transformations. These include total forward packets, total number of push flags in the forward direction, maximum packet interarrival time (IAT) in the forward direction, etc.
(3) Features whose values depend on the second group and can be calculated directly by a set of them.
(4) Features that cannot be directly recalculated based on independent features, and a sequence of packets affect their values.

We tailored our adversarial crafting algorithm based on these 4 groups. We defined 3 masks that are the subset of each other. Each mask blocks a specific numbers of features from being updated by back propagating gradients through the models. The first mask only allows the procedure to modify the independent features (e.g., the second group). The second mask adds some of the 4th group features, and finally, the third mask adds all of the features of the fourth group to the set of modifiable features. In the crafting procedure, we first check whether we can fool the NIDS by using

the first mask. In the case of failure we use the second and third masks. More specifically, the loss function we defined to minimize during the crafting procedure is as follows:

$$AdvLoss = F(x + \delta \odot mask_i)$$

where F is the model and $F(.)$ is the score predicted by the model. $\odot$ is the element-wise multiplication operator and δ is the perturbation that we want to find to add to the original features to fool the NIDS. By generating the adversarial features this way, we can be sure that applying legitimate transformations to the malicious flows will result in each feature from the first three groups matching the adversarial feature found.

However, the fourth group of features would have different values, and that can cause the overall flow to be detected by the NIDS. Therefore, in order to increase the chance of fooling the NIDS, in the crafting procedure, we do not stop the algorithm immediately after the score of a given sample drops below the threshold. To have a confidence interval, we continue to modify features in order to decrease the score further below the threshold. We considered this interval in order to compensate the effect of different values between the fourth group of features and increase the chance of fooling the NIDS with the real sequence of packets.

Algorithm 2 demonstrates how we tailored the crafting procedure for flow-based NIDSs. In this algorithm $threshold'$ is a smaller value than the real threshold of the NIDS to provide the confidence interval we discussed. Note that we start with a small learning rate to keep the modifications small and increase the learning rate exponentially in case of failure. The adversarial features we find with this algorithm against a given NIDS show the lower bound of the NIDSs robustness. This is because for some of the adversarial examples, there might not be a real sequence of packets that have those features.

Algorithm 2 Crating adversarial examples for Flow-based NIDSs

```
procedure CraftAdvEx(x)                          ▷ x is a malicious flow
    for each mask ∈ mask_1, mask_2, mask_3 do
        for each lr ∈ 0.001, 0.01, 0.1, 1.0 do
            for each i ∈ [0, totalIter] do
                take one step of GD with learning rate=lr
                x' ← x + δ
                Recalculate group 3 features
                if score_x' < threshold' then
                    return x'
                end if
            end for
        end for
    end for
end procedure
```

5 EVALUATION

In this section, we evaluate the performance of the aforementioned NIDSs in both a normal setting and an adversarial setting with the traffic manipulations described in Section 3. We first discuss the dataset used, then discuss the metrics used for our evaluation and finally, empirically demonstrate to what degree Algorithms 1 and 2 are effective in fooling different NIDSs.

5.1 Dataset

To evaluate network intrusion detection systems, we used a highly cited dataset containing network traces of twelve network attacks from the Canadian Institute of Cybersecurity (CIC) [1] [22]. Sharafaldin et al. in [22] compared eleven available datasets based on eleven criteria and concluded that all of them have some shortages such as lack of traffic diversity and volumes, limited number of attacks, etc. Therefore they built a new dataset which satisfies all of the eleven criteria.

The attacks are: FTP-Patator, SSH-Patator, Dos slowloris, DoS slowhttptest, DoS Hulk, DoS GoldenEye, Heartbleed, Web attacks, Infilitration, Botnet, PortScan and DDoS. These attacks were carried out over a 5-day work week in a controlled environment. Each attack was implemented using popular network tools or was written in Python by the authors. The network traces of each attack were collected to study and identify intrusion traffic characteristics.

The CICIDS2017 [22] dataset contains flows extracted from packets files using the CICFlowMeter Tool [11]. The tool also extracts 80 behavioral flow features for each flow. The full list of features can be seen in the Appendix in Table 1. Each flow and its corresponding flow features were labeled as either benign or with the specific attack name, but the individual packets were not labeled. Thus, in order to evaluate the packet based NIDSs, we labeled packets as malicious or benign based on the information Sharafaldin et al. provided for this dataset. From the PCAP files provided within the dataset, we excluded IPv6 packets and labeled the other packets in the following way: for each attack, we labeled all of the packets sent or received between the attacker IP(s) and the victim IP(s) as malicious for the duration of that attack. All other packets were labeled as benign. We also exclude web attacks from our evaluation because the features extracted in our evaluation are only from packet headers and detecting web attacks requires deep packet inspection. The whole dataset contains more than 56 million packets. We trained the packet and flow-based NIDSs on the Monday traffic, which contains over 11.6 million benign packets (529,481 flows). The NIDSs were then tested on the network traffic generated from Tuesday to Friday, which contains both benign and network attack traffic. This test set contains 12 different network attacks, which make up 10.33% of the overall packets and 24.22% of the overall flows. The dataset's full packet and flow statistics can be found in Table 2 in the Appendix.

5.2 Evaluation Metrics

5.2.1 True Positive Rate (TPR). TPR shows the ratio of malicious traffic that is detected as malicious to all of the malicious traffic when the model's threshold is fixed to a specific number.

5.2.2 False Positive Rate (FPR). FPR shows the ratio of benign traffic that is considered malicious to all of the benign traffic when the model's threshold is fixed to a specific number.

5.3 Performance in Adversarial Setting

In order to see how each of the aforementioned NIDSs detect adversarially modified network attacks, we chose their individual thresholds in a way to keep their FPR at 0.1 since those NIDSs

[1] The dataset can be downloaded at: https://www.unb.ca/cic/datasets/ids-2017.html

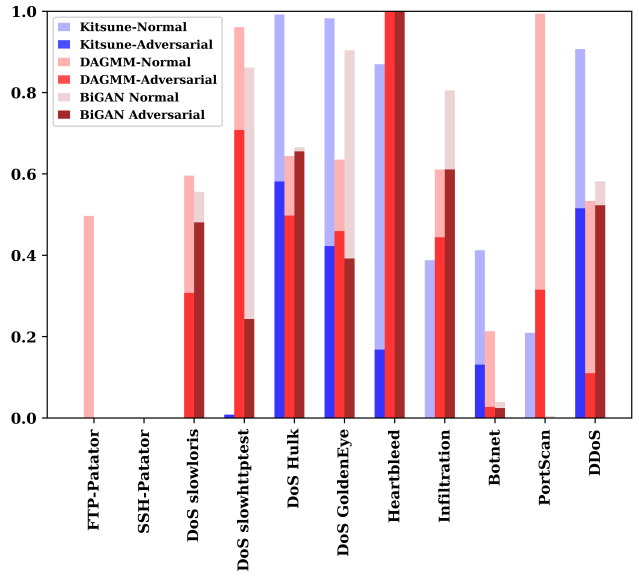

Figure 2: The TPR of different NIDSs for each attack when FPR is 0.1 when sending normal traffic and the adversarial version of it.

can detect most of the network attacks at this rate in a normal setting. In order to evaluate Kitsune, we used GMM as its detector because it could detect malicious traffic better than using the suggested ensemble of autoencoders. To fool this NIDS, we modified the malicious packets from the CICIDS2017 dataset with Algorithm 1. We fed all the packets into the NIDS, as in the normal setting, but due to the computational complexity of crafting adversarial examples, we only ran it on the first 25,000 packets of an attack. To evaluate the flow-based NIDSs in an adversarial setting, we used Algorithm 2 to find the adversarial features for malicious flows. Due to the computational complexity of this procedure we only did it for the first 5000 flows of each attack in the cases where the attack contained more than 5000 flows.

The results of this evaluation are shown in Figure 2. For each NIDS considered, we show both the TPR under normal conditions, as well as under adversarial conditions. As it can be seen in this figure, for a packet-based NIDS, the detection rate drops by up to 70% (for Heartblead) in adversarial setting and for flow-based NIDSs the detection rate drops by up to 68% (for PortScan). In fact, the performance of each NIDS decreases dramatically in most cases, indicating that these NIDS are not robust in the face of adversarial examples. More specifically, for Kitsune, the average TPR in an adversarial setting across all attacks drops to 16.6% from 43.6% in a normal setting; for DAGMM, it drops to 35.2% from 60.8%, and for BiGAN-based, it drops to 35.7% from 49.3%.

6 DISCUSSION AND FUTURE WORKS

As we saw, all of the previous NIDSs are vulnerable to adversarial example attacks to some degree. One thing that is common among all of them is that they behave in a deterministic way. That is to say, the inputs that the attacker generates by the help of her local copy will fool the victim's NIDS if they fool the local copy; for the same input these NIDSs output the same score. We know that the attacker can't query the victim's NIDS directly otherwise she would be detected. Therefore, one way to make NIDSs more robust against adversarial examples is to make them behave stochastically. Then for the same input the adversary's local copy and actual NIDS would output different scores. This adds extra hardship for making adversarial examples. Because in this case, fooling the local copy doesn't guarantee to fool the actual NIDS. In the future, we want to design an NIDS by this approach to see how it improves the robustness of the NIDS against adversarial examples.

Also, as we mentioned earlier, for the evaluation of flow-based NIDSs, we found the lower bound of robustness of each NIDS. In order to find their exact robustness, we need a tool to modify packets in a flow to match the extracted features with those we found. We leave production of this tool for future work.

Also in previous work [12], it is discussed how to add a network forwarding element known as a traffic normalizer on the joint path of the NIDSs and the victims to make sure that the NIDS processes the same packets that the victim does. Thereby, limiting the ability of an attacker outside the victim's network to inject fake packets into the stream by using bad sequence numbers, specific TTL values, etc. However, noramlizers tend to introduce complications for normal traffic. A normalizer can raise the TTL value of packets it forwards to a pre-defined value to ensure the packet is not only processed by the NIDS, but also by the destination. This can cause some of the packets to loop forever and consume a network's bandwidth if the normalizer is deployed on a loop. Also, normalizers can prevents some of the tools that are used for debugging in the network from functioning properly, such as traceroute. Furthermore, normalizers process each packet in depth; therefore, adding latency into communication channels. Extra latency can prove to be detrimental to real-time applications. Due to the issues these normalizers introduce, they are not widely used, so we did not consider them in our evaluations. But even by considering them, they won't affect the algorithms we introduced to modify a traffic. The only difference is that only the delay and split transformations can be used to fool the NIDS in this situation.

7 CONCLUSION

In this paper, we showed how to evaluate anomaly-based NIDSs in an adversarial setting. We identified the legitimate transformations which an adversary can make to malicious traffic to fool the NIDS and not break the underlying network protocols. Finally, in our Empirical study, we showed the effectiveness of our approach by tailoring the legitimate transformations towards both packet-based and flow-based NIDSs. We found out that by using the transformations we introduced in this paper the detection rate of an NIDS trained on packet-level features can be dropped by up to 70% and the detection rate of an NIDS trained on flow-level features can be dropped by up to 68%.

ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation (NSF) (Award Number 1406192) and VMware and NSF (Award

Number 1700527). We also thank NVIDIA Corporation for the donation of the Titan V GPU used for this research.

REFERENCES

[1] M. Al-Qatf, Y. Lasheng, M. Al-Habib, and K. Al-Sabahi. 2018. Deep Learning Approach Combining Sparse Autoencoder With SVM for Network Intrusion Detection. *IEEE Access* 6 (2018), 52843–52856. https://doi.org/10.1109/ACCESS.2018.2869577

[2] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*. https://arxiv.org/abs/1802.00420

[3] Shumeet Baluja and Ian Fischer. 2018. Learning to Attack: Adversarial Transformation Networks. In *Proceedings of AAAI-2018*. http://www.esprockets.com/papers/aaai2018.pdf

[4] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks against Machine Learning at Test Time. In *ECML/PKDD*.

[5] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy*. 39–57.

[6] Nicholas Carlini and David Wagner. 2018. Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. In *Deep Learning and Security Workshop*.

[7] Raghavendra Chalapathy and Sanjay Chawla. 2019. Deep Learning for Anomaly Detection: A Survey. *CoRR* abs/1901.03407 (2019). arXiv:1901.03407 http://arxiv.org/abs/1901.03407

[8] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. 2018. Robust Physical-World Attacks on Deep Learning Visual Classification. In *Computer Vision and Pattern Recognition*. IEEE.

[9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

[10] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial Examples for Malware Detection. In *Computer Security – ESORICS 2017*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer International Publishing, Cham, 62–79.

[11] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. 2016. Characterization of Encrypted and VPN Traffic Using Time-Related Features. https://doi.org/10.5220/0005740704070414

[12] Mark Handley, Vern Paxson, and Christian Kreibich. 2001. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *USENIX Security Symposium*.

[13] Mohammad Hashemi, Greg Cusack, and Eric Keller. 2018. Stochastic Substitute Training: A Gray-box Approach to Craft Adversarial Examples Against Gradient Obfuscation Defenses. In *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security (AISec '18)*. ACM, New York, NY, USA, 25–36. https://doi.org/10.1145/3270101.3270111

[14] ICS-CERT. 2016. Cyber-attack against Ukrainian critical infrastructure. www.ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01.

[15] Ram Shankar Siva Kumar, Andrew Wicker, and Matt Swann. 2017. Practical Machine Learning for Cloud Intrusion Detection: Challenges and the Way Forward. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec '17)*. ACM, New York, NY, USA, 81–90. https://doi.org/10.1145/3128572.3140445

[16] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2017. Delving into Transferable Adversarial Examples and Black-box Attacks. In *International Conference on Learning Representations*.

[17] R. K. Malaiya, D. Kwon, J. Kim, S. C. Suh, H. Kim, and I. Kim. 2018. An Empirical Evaluation of Deep Learning for Network Anomaly Detection. In *2018 International Conference on Computing, Networking and Communications (ICNC)*. 893–898. https://doi.org/10.1109/ICCNC.2018.8390278

[18] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In *Network and Distributed System Security Symposium 2018 (NDSS'18)*.

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

[20] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks Against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS '17)*. ACM, New York, NY, USA, 506–519. https://doi.org/10.1145/3052973.3053009

[21] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 372–387.

[22] Iman Sharafaldin, Arash Habibi Lashkari, , and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *4th International Conference on Information Systems Security and Privacy (ICISSP)*.

[23] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. 2016. 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)*. ACM, New York, NY, USA, 1528–1540. https://doi.org/10.1145/2976749.2978392

[24] Symantec. 2019. Internet Security Threat Report. https://www.symantec.com/security-center/threat-report.

[25] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[26] C. Yin, Y. Zhu, S. Liu, J. Fei, and H. Zhang. 2018. An enhancing framework for botnet detection using generative adversarial networks. In *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*. 228–234. https://doi.org/10.1109/ICAIBD.2018.8396200

[27] Yang Yu, Jun Long, and Zhiping Cai. 2017. Network Intrusion Detection through Stacking Dilated Convolutional Autoencoders. *Security and Communication Networks* 2017 (2017). https://doi.org/10.1155/2017/4184196

[28] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. 2018. Efficient GAN-Based Anomaly Detection. *CoRR* abs/1802.06222 (2018). arXiv:1802.06222 http://arxiv.org/abs/1802.06222

[29] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJJLHbb0-

APPENDIX

The details of the dataset we used for our evaluation.

Features	Fwd	Bwd	Flow
Total Duration	✗	✗	✓
Total Packets	✓	✓	✗
Total Length of Packets	✓	✓	✗
Pkt Len Min/Max/Mean/Stddev	✓	✓	✓
IAT Min/Max/Mean/Sttdev	✓	✓	✓
Bytes/s	✗	✗	✓
Pkts/s	✓	✓	✓
PSH/URG Flags	✓	✓	✓
FIN/SYN/RST/ACK/CWE/ECE Flags	✗	✗	✓
Total Length of Headers	✓	✓	✗
Down/Up Ratio	N/A	N/A	✓
Avg Bytes/Bulk	✓	✓	✗
Avg Packets/Bulk	✓	✓	✗
Avg Bulk Rate	✓	✓	✗
Initial Window Bytes	✓	✓	N/A
Packets w/ payload >= 1	✓	✗	✗
Min. Packet Header Size	✗	✓	✗
Active Time Min/Max/Mean/Stddev	✗	✗	✓
Idle Time Min/Max/Mean/Stddev	✗	✗	✓

Table 1: Features extracted from flows for classifying network traffic with flow-based NIDS. ✓ and ✗ indicate whether or not the feature was calculated for packets in moving in the labeled direction. "Flow" indicates features calculated taking into account packets flowing in both directions. Features were extracted using the CICFlowMeter Tool [11].

Set	Type	# of P	% of P	# of F	% of F
Train	Benign	11,680,917	100	529,481	100
Test	Benign	39,946,287	89.67	1,741,803	75.78
	FTP-Patator	110,736	0.25	7,935	0.35
	SSH-Patator	138,621	0.31	5,897	0.26
	DoS slowloris	47,586	0.11	5,796	0.25
	DoS slowhttptest	39,257	0.09	5,499	0.24
	DoS Hulk	2,245,526	5.04	230,124	10.01
	DoS GoldenEye	106,177	0.24	10,293	0.45
	Heartbleed	49,296	0.11	11	0.00
	Web Atks	39,823	0.09	2,179	0.10
	Infiltration	209,920	0.47	36	0.00
	Botnet	9,871	0.02	1,956	0.09
	PortScan	324,062	0.73	158,839	6.91
	DDoS	1,280,602	2.87	128,025	5.57
	All Attacks	4,601,477	10.33	556,628	24.22
	All	44,547,764	100.00	2,298,431	100.00

Table 2: The statistics of the dataset used for our evaluation. Columns headers containing "P" contain packet information, while column headers containing "F" show flow information.