Stealthy Network Attacks that can Evade Neural Network Based Detectors

Qingtian Zou [a,*], Lan Zhang [a], Anoop Singhal [b], Xiaoyan Sun [c] and Peng Liu [a]
[a] *College of Information Sciences and Technology, The Pennsylvania State University, PA, USA*
E-mails: qzz32@psu.edu, lfz5092@psu.edu, pxl20@psu.edu
[b] *Security Test, Validation and Measurement Group, National Institute of Standards and Technology, MD, USA*
E-mail: anoop.singhal@nist.gov
[c] *College of Engineering & Computer Science, California State University, Sacramento, CA, USA*
E-mail: xiaoyan.sun@csus.edu

Abstract. While network attacks play a critical role in many advanced persistent threat (APT) campaigns, an arms race exists between the network defenders and the adversary: to make APT campaigns stealthy, the adversary is strongly motivated to evade the detection system. However, new studies have shown that neural network is likely a game-changer in the arms race: neural network could be applied to achieve accurate, signature-free, and low-false-alarm-rate detection. In this work, we investigate whether the adversary could fight back during the next phase of the arms race. In particular, noticing that none of the existing adversarial example generation methods could generate malicious packets (and sessions) that can simultaneously compromise the target machine and evade the neural network detection model, we propose a novel attack method to achieve this goal. We have designed and implemented the new attack. We have also used Address Resolution Protocol (ARP) Poisoning and Domain Name System (DNS) Cache Poisoning as the case study to demonstrate the effectiveness of the proposed attack.

Keywords: Network Attack, Neural Network, Adversarial Example

1. Introduction

Network intrusion detection systems (NIDS), such as Snort [1] and Zeek [2], play an essential role in enterprise network security operations. With the rapid rise of deep learning technology, neural network(NN)-based detectors (e.g., [3–6]) have been increasingly deployed in the real world to complement the existing NIDS. Acting as an “ally", NN-based detectors enable accurate detection of attacks even those not seen earlier, before the corresponding intrusion detection rule is created. NN-based detectors are usually deployed in the real world as follows: the existing network traffic monitoring tools (e.g., routers, Wireshark packet sniffers) are reused to capture the *captured packets*, which are usually held in PCAP files, to a central node (e.g., a node in a cloud) where the NN-based detectors are running with machine learning libraries (e.g, PyTorch, TensorFlow).

Because adversarial examples [7, 8] are widely recognized as a most serious threat to systems that use machine learning models, NN-based NIDS have been triggering an increasing amount of interests in the research community. In particular, several works (e.g., [9, 10]) have recently investigated the

*Corresponding author. E-mail: qzz32@psu.edu.

adversarial examples specific to NN-based NIDS, and they assume the following threat model: **The data manipulation threat model.** This threat model assumes that the adversary is able to *directly manipulate (i.e. modify)* the network data after it is captured (e.g., packets held in PCAP files) and/or processed (e.g., data samples), but before it is fed to the NN-based NIDS.

Although this threat model is coherent with the adversarial example studies [7, 11] in other domains such as computer vision, it is **very different** from the threat model assumed by real-world NIDS. When NIDS are deployed to defend against network attacks, the following threat model is assumed as a matter of fact: **The de facto standard threat model**. This threat model does NOT assume that the adversary is able to manipulate data after it is generated. Rather, this threat model assumes that (a) the adversary has compromised a non-admin user account and can run attack scripts on an internal machine utilizing the account's permission; (b) the adversary can sniff certain kinds of network packets. Most real-world network attacks (e.g., DNS cache poisoning, Pass the Hash) follow the standard threat model.

Regarding why the data manipulation threat model is not assumed by real-world NIDS, we have the following observations. First, most real-world network attacks only compromise non-admin user accounts, and such accounts do not have the permission to modify the packets held in packet capture repositories. Second, packet capture repositories are usually guarded by strict access control and integrity protection. Countering such access control and integrity protection assumes additional capabilities of the attackers.

Based on the above observations, it is important to check whether the adversarial examples successfully generated under the data manipulation threat model are STILL successful under the de facto standard threat model. By "STILL successful", we mean the following attack procedure: (a) the adversary treats the adversarial examples generated by the existing methods as an adversarial attack packet sent out from the adversary's attack script, or the adversarial examples can be somehow converted to adversarial attack packets; (b) when the attack script is running, the adversarial attack packets that are directly sent to the target machine can STILL compromise the target machine; (c) the network data during this time, after collected and processed in the same way as what is required of the NN-based NIDS, is STILL classified as benign.

Using the above attack procedure, we have checked 17 general-purpose adversarial-example generation methods. We will summarize the experiment results in Section 2. In short, we found that none of the adversarial examples generated in our experiments is still successful under the standard threat model.

In this work, we seek to develop the **first** approach which can generate successful adversarial examples under the de facto standard threat model. These successful adversarial examples are denoted as *stealthy network attacks* in this paper, since the proposed attack and the real-world network attacks follow the same threat model.

The main challenges we must address in developing the new approach are rooted in why the existing methods fail to handle the standard threat model. Taking the Address Resolution Protocol (ARP) poisoning attack for example, since the attack script can poison the target machine's ARP table through one packet, each adversarial example for the NN-based detector is a single packet. When such an attack packet was perturbed by 17 general-purpose adversarial example generation methods, which we will enumerate shortly in Section 2, we observed that although the generated adversarial examples may evade the NN-based detector under the MCP threat model, they all failed to poison the target machine's ARP table. A main reason is that the perturbed packets violate some particular network **protocol constraints** (e.g., the values for a packet field should be fixed; the values for different packet fields may be correlated). When some protocol constraints are violated, the perturbed packets are usually treated as

invalid packets and discarded by the target machine. In addition, we found that when detecting multi-packet, session-based attacks (e.g., DNS cache poisoning attack), the corresponding adversarial examples are more likely to violate some protocol constraints (e.g., the values of one field in packet A and the values of another field in packet B may be correlated). Because network protocol constraints are domain-specific, general-purpose adversarial example generation methods are unlikely to automatically satisfy such constraints. In sum, how to generate protocol-constraint-satisfying adversarial examples under the standard threat model is the primary challenge we must address.

To the best of our knowledge, this work is the **first** work addressing this challenge and actually test the effectiveness of adversarial attack packets in real attacks. Our approach firstly uses masks to capture protocol constraints. Then, our approach adapts the existing adversarial example generation methods by applying masks to them. In particular, we make the following two main modifications: 1) we apply masks when generating perturbations; 2) when generating perturbed data samples, we confine and round all values to the valid range specified for each field. With these two modifications, our approach can generate **constraint-satisfying** adversarial examples. For multi-packet network attacks, our approach uses a record-and-replay mechanism to help address the session-level uncertainties that the adversary may face when performing stealthy attacks in a real network.

The main contributions of this work are as follows: 1) We developed the first approach which can generate successful adversarial examples under the de facto standard threat model. The adversary can use these adversarial examples to launch stealthy network attacks and evade NN-based NIDS. 2) We used ARP poisoning and DNS cache poisoning attacks as the case studies to demonstrate the effectiveness of the proposed approach.

The rest of the paper is organized as follows. Section 2 provides the background and highlights the motivation. Section 3 describes the NN-based detectors. Section 4 formally defines constraint-satisfying adversarial examples, and then presents our problem statement. Section 5 described our proposed approach. Section 6 presents the evaluation results of the proposed approach. Section 7 discusses several relevant issues. Section 8 summarizes the related work. Section 9 concludes the entire paper.

2. Background and Motivation

2.1. Network Attacks and NIDS

Network attacks are frequently used in Advanced Persistent Threats (APTs). Some common network attack types include probing, denial of service (DoS), Remote-to-local (R2L), etc. These attacks can often cause serious impacts. For example, ARP poisoning and DNS cache poisoning are two commonly seen network attacks belonging to R2L attack type. With ARP poisoning attack, attackers can intercept or redirect network traffics within the LAN to any desired MAC addresses. In DNS cache poisoning attack, a falsified DNS record with the fake domain-to-IP mapping will be added and persist in the local DNS server until it expires. Whenever a machine inquires the poisoned DNS server for the affected domain's IP, it will get the fake IP address.

Signature-based, rule-based, and anomaly-based detection are commonly used to detect these network attacks. However, signature-based methods [12] can be easily fooled by slightly changing the attack payload; rule-based methods [1] need experts to formulate and regularly update rules; and anomaly based detection [13] tends to raise lots of false positives. Especially in attacks where attackers can intentionally craft their packets to make them seem legitimate, such as in ARP poisoning and DNS cache poisoning

attacks, the aforementioned methods usually fall short. Therefore, new detection methods need to be explored.

2.2. NN-based Network Attack Detectors

Recently, researchers have been applying machine learning and deep learning techniques for network attack detection. DeepDefense [14], applying recurrent neural networks (RNN) for DoS detection, achieves higher than 97% accuracy; PCCN [4] which uses convolutional neural networks (CNN) to detect abnormal network traffic flows, can achieve higher than 99% accuracy; Long-Short Term Memory (LSTM) neural network has also demonstrated an accuracy of higher than 99% for detecting network attacks [5]. Many industries, including Splunk [15], FireEye [16], and Fortinet [17], etc., are incorporating deep learning into their network security products. One recent work [6] also studies applying deep learning techniques to the detection of two network attacks: ARP poisoning and DNS cache poisoning. The best trained models are quite accurate: for ARP poisoning, the detection rate is about 99.8%; for DNS cache poisoning, the detection rate is 100%. In another word, evasion rates are about $1 - 99.8\% = 0.2\%$ for ARP poisoning, and 0% for DNS cache poisoning. Therefore, **blindly launching network attacks in the same old fashion without any countermeasures to evade neural networks is very likely to be detected.** This will directly motivate attackers to alter their ways of launching network attacks.

2.3. Adversarial examples for DNNs

Adversarial examples are are specialised data examples created with the purpose of confusing a DNN, resulting in the misclassification of a given data example. In the domain of computer vision, these adversarial data examples are indistinguishable to the human eye, but cause the DNN to fail to classify the corresponding images. In the domain of network security, these adversarial data examples cause the DNN to fail to classify the corresponding network data. A variety of automatic adversarial example generation methods have been proposed in the literature, and 17 of these methods will be shortly summarized in the next section.

2.4. Why the adversarial examples generated by existing methods fail under the standard threat model

Our experiments. We tested 17 general-purpose adversarial example generation methods against the DNN trained to detect ARP poisoning attacks. (When we did the experiments, the 17 methods were all the methods that we are aware of.) In particular, we followed the attack procedure described in Section 1, that is, we firstly used the 17 methods to generate a set of adversarial examples against the DNN; then we used the generated adversarial examples to get the adversarial attack packets; finally, we ran the attack scripts (under the standard threat model) to directly send the attack packets to the target machine. After the attack packets were sent out, we used Wireshark to monitor/capture these packets, and the captured packets are fed into the DNN after the required data pre-processing.

The results are summarized in Table 1. The third column shows that 9 out of the 17 methods generated one or more adversarial examples, and in total more than 500 adversarial examples were generated. Unfortunately, the last column shows that none of the adversarial examples successfully compromised the target machine.

Why the existing methods failed. One reason is that the network packets also need to follow certain network protocol constraints (e.g. requiring certain values in specific fields). Another reason is that the attacker may only have limited control over the target network, and he/she cannot modify packets

Table 1
General purpose adversarial example generation methods fail when launching the ARP poisoning attack.

Adv. Ex. Gen. methods	Time (s)	Number of generated adversarial examples	Number of successful ARP poisoning attacks using generated adversarial examples
FGSM [7]	0.02	0	0
BIM [19]	1.99	0	0
DeepFool [20]	0.73	98	0
JSMA [21]	0.51	0	0
VAT [22]	44.40	0	0
Carlini [23]	79.02	1	0
Inv. [24]	0.01	100	0
NewtonFool [25]	20.25	0	0
ZOO [26]	144.92	100	0
Boundary [27]	280.77	0	0
EAD [18]	369.10	89	0
PGD [28]	0.33	1	0
BBA [29]	33.44	0	0
DDNA [30]	0.57	5	0
Gen [31]	688.50	98	0
SLIDE [32]	0.33	0	0
HSJA [33]	2273.94	96	0

sent out from other machines. The existence of these two issues means that the adversarial attack packets can result in modifications of only a portion of a data sample fed into the NN-based detector, and the modifications need to satisfy certain constraints. For example, we examined the adversarial attack packets back-converted from the adversarial examples generated by the adversarial example generation method proposed in [18]. We found that majority of these packets are not recognized as ARP packets, because the value of the *type* field in the data link layer has been perturbed and is not ARP (0x0806). Although a small portion of the back-converted packets are correctly recognized as ARP packets (*type* field has value 0x0806), the values of the hardware type (*HTYPE*) field, the protocol type (*PTYPE*) field, or the hardware length (*HLEN*) field have been changed. However, in order to satisfy the ARP protocol constraints, none of these values should be changed; otherwise, the corresponding packets may be abandoned by the recipient.

3. Dataset and Detection Neural Networks

Before presenting our attack scheme, we will first introduce the detection neural network and the dataset on which the models are trained. In this work, we focus on the logic-flaw-exploiting (LFE) network attacks [6], which exploit the logic (security) flaws of a few widely-deployed authentication protocols. Such attacks are very different from other attacks such as memory corruption for code reusing, command and control (C&C) over HTTP/HTTPS, and (distributed) denial of service with/without botnet, etc. Memory corruption is more about the server program rather than the network protocol; C&C over HTTP/HTTPS assume the HTTP/HTTPS protocol itself is running normally; and (distributed) denial of service is usually accomplished by exhausting the server's resources instead of exploiting a logic flaw within a protocol.

Table 2
Dataset statistics.

Attacks	Set	Size	Benign to malicious ratio
ARP poisoning	Training	9584	1.005:1
	Test	2400	0.982:1
DNS cache poisoning	Training	30928	1.003:1
	Test	7732	0.988:1

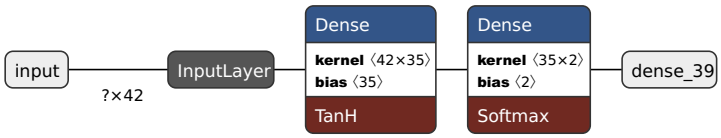


Fig. 1. MLP neural network for ARP poisoning detection. Visualization by Netron[1].

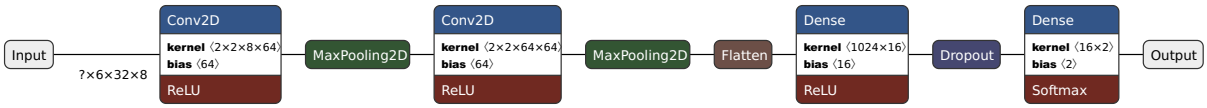


Fig. 2. CNN for DNS cache poisoning detection. Visualization by Netron[1].

Two representative LFE network attacks are address resolution protocol (ARP) poisoning attacks and domain name system (DNS) cache poisoning attacks. ARP poisoning works by spoofing ARP responses, with which the attacker can trick the victim into falsified mappings between IP addresses and MAC addresses, and thus intervening the network communication. DNS poisoning works by spoofing DNS responses, with which the attacker can trick the victim into falsified mappings between domain names and IP address, and thus redirecting the network communication. Both of these two attacks exploit the lack of response verification in the corresponding protocol. As a result, the victims cannot verify whether the packets comes from a genuine host or attacker. Also, these two attacks are difficult to be detected with traditional detection methods (e.g. signatures, rules, anomaly detections, etc.) because spoofing is applied (i.e. attacker packets are intentionally crafted to be indistinguishable from normal packets).

The detection neural networks and datasets are from another work [6], in which self-generated network datasets are generated due to lack of public datasets on the two attacks. Dataset statistics are shown in Table 2.

For ARP poisoning detection, a multi-layer perceptron (MLP) neural network is used, as shown in Fig. 1. Every data sample represents one network packet. 42 bytes are selected from each ARP packet. Every byte is then treated as a number when constructing data samples. Hence, each data sample for ARP poisoning is a 1D vector containing 42 integers.

For DNS cache poisoning detection, a convolutional neural network (CNN) is used, as shown in Fig. 2. The CNN's input is 3D matrices. Firstly, starting from a whole network log, we filter out the DNS packets and chop them by sessions, resulting in multiple variate-length DNS sessions. Secondly, variate-length DNS sessions are chopped into fixed-length slices by applying a sliding window of size 6. Thirdly, in every DNS packet, 32 bytes from `IP` layer, `UDP` layer, and `DNS` layer are chosen. Each byte is converted to 8 bits, and then treated as an integer (0 or 1). As a result, every data sample for DNS cache poisoning is a 3D matrix of shape $6 * 32 * 8$.

Table 3
Test set evaluation results.

Attack	ARP poisoning	DNS cache poisoning
Model architecture	MLP	CNN
Accuracy	99.75%	99.73%
F1 score	0.9975	0.9973
Detection rate	99.59%	99.53%
False positive rate	0.08%	0.08%

The two models' performances when detecting real attacks are summarized in Table 3. (Please refer to the original work for more details.) It can be inferred that, if the attacker takes no counter-measurements about the neural network detection, the attack is very likely to get detected.

4. Problem Formulation

4.1. Protocol-Constraint-AWare Adversarial Examples

In this work, we first present a simple scenario, in which ARP poisoning is used as a running example for demonstration throughout the paper. The attack is completed with a single ARP packet, and the detection is also based on individual ARP packets.

As defined in Section 1, the attacker's goal is to launch stealthy network attacks, which are accomplished by **Protocol-Constraint-AWare (PCAW) adversarial examples**. PCAW adversarial examples refer to the data samples that satisfy stealthy network attacks. PCAW adversarial examples have two intrinsic aspects: they are protocol-constraint-aware, meaning that they follow the protocol constraints; and they are adversarial examples, meaning that neural network detection models misclassify them. In ARP poisoning, they refer to data samples (generated from ARP packets) which can be used to launch the ARP poisoning attack, and are also misclassified by the detection model as benign.

A stealthy network attack contains at least two phases: 1) generating *PCAW adversarial examples*; 2) launching the stealthy network attack with the data from generated PCAW adversarial examples. The network packets that can lead to successful stealthy network attacks are called **stealthy attack packets**.

Constructing PCAW adversarial examples for attacks in phase 1 is much more challenging than constructing adversarial examples in computer vision: 1) Network communications must follow certain protocols; 2) The attack packets must have certain bytes so that these packets are valid and can exploit the vulnerability in the target machine/service.

Therefore, the following two challenges should be addressed:

A. How should the PCAW adversarial examples be generated for given neural network models?
B. How to launch stealthy network attacks with the help of generated PCAW adversarial examples?

For demonstration, we first use ARP poisoning attack [6]. The ARP poisoning attack is selected because: 1) The attacks can be detected largely, if not solely, based on network logs. Thus, neural network detection based on network logs is made possible. 2) The data samples of those detection neural networks present bytes from network packets, so the mapping between features in data samples and bytes in packets is straightforward. 3) There are no distinctive signatures for detecting these attacks. Thus,

[1] https://github.com/lutzroeder/netron

traditional detecting methods often fall short and deep learning can be applied. 4) The ARP poisoning attack represents single-packet network attacks, which is simpler than DNS cache poisoning, the multi-packet network attacks.

4.2. Unique Challenges in Multi-packet Stealthy Network Attacks

Extending the proposed stealthy attack packet generation method to the multi-packet network attack scenario has some unique challenges.

Assuming that one multi-packet attack session consists of eight packets $S = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$, where packets x_3 and x_7 are attacker's packets. A common data processing technique to deal with this kind of data is sliding window. Assuming that a sliding window of window size 6 and step size 1 is applied, the session above is processed into three data samples denoted as $s_1 = [x_1, x_2, x_3, x_4, x_5, x_6]$, $s_2 = [x_2, x_3, x_4, x_5, x_6, x_7]$, and $s_3 = [x_3, x_4, x_5, x_6, x_7, x_8]$. All these data samples are generated from the attack session and are therefore labeled as malicious. The session S will be detected as an attack session if any of the data samples are classified as malicious by the neural networks. Therefore, to evade detection, all data samples should be misclassified as benign. Hence, the ideal solution for attackers is to modify $x_3 \rightarrow x'_3$ and $x_7 \rightarrow x'_7$, so that $s'_1 = [x_1, x_2, x'_3, x_4, x_5, x_6]$, $s'_2 = [x_2, x'_3, x_4, x_5, x_6, x'_7]$, and $s'_3 = [x'_3, x_4, x_5, x_6, x'_7, x_8]$ can be misclassified as benign, and meanwhile x'_3 and x'_7 can still lead to successful attacks. Consequently, $S' = [x_1, x_2, x'_3, x_4, x_5, x_6, x'_7, x_8]$ will not be detected as an attack session by the neural networks, although it is actually malicious. To align with our definition, we call such attack sessions, S', as **stealthy network attack sessions**: they can compromise the network, similar to any other network attacks, and also remain stealthy by evading the detection of neural network models.

First of all, without compromising other machines, attackers have no control over packets sent by other machines. In another word, attackers can only modify their own attack packets, which usually results in a small portion of the data sample(s). As a result, generating PCAW adversarial examples will be difficult for such data sample(s) because only a small portion can be perturbed. In the example above, $s_1 = [x_1, x_2, x_3, x_4, x_5, x_6]$ has only one attack packet (x_3), which means the attacker can only modify this $x_3 \rightarrow x'_3$ to make $s'_1 = [x_1, x_2, x'_3, x_4, x_5, x_6]$ a PCAW adversarial example. Not surprisingly this may have a low chance of success.

Apart from the difficulty of generating PCAW adversarial examples, stealthy multi-packet attacks also have challenges that stem from the nature of sessions. In multi-packet attacks and detections, one session usually corresponds to one or more data samples. In a malicious session, an attacker's packet may appear in one or more data samples from this session, depending on how the session is processed into data samples. For example, in the aforementioned example, the attacker's packet x_3 in session S is involved in three different data samples s_1, s_2, and s_3. This creates some **uncertainties** for launching stealthy network attacks:

(1) When generating adversarial examples from different data samples (e.g. s_1, s_2) that contain the same packet (e.g. x_3), information related to this packet might be modified in different ways. It is difficult to accommodate and reflect the different modifications in just one packet.
(2) Some data samples, which belong to the same session, might fail to produce adversarial examples. For example, s_1 might generate an adversarial example, which can be converted reversely to network packets including packet x'_3, but s_2 might fail to generate adversarial examples, meaning no packet x'_3 can make this data sample misclassified.
(3) Even if an attack packet x'_3 can make all the three data samples misclassified, this x'_3 might only work within this session S. Whether it is effective in other sessions is uncertain.

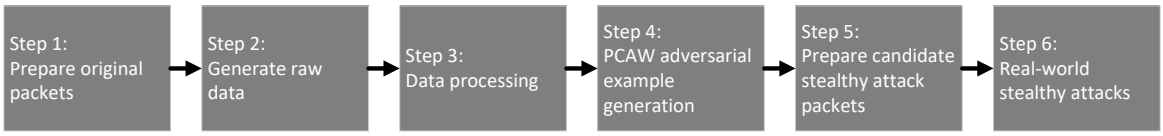


Fig. 3. The work flow of launching stealthy network attacks.

All these uncertainties should be mitigated after PCAW adversarial examples are generated, but before attacks are launched in real world.

5. Proposed Stealthy Adversarial Attacks

5.1. Work Flow of Stealthy Network Attacks

The work flow (see Figure 3) of the proposed stealthy network attacks is as follows:

(1) Attackers prepare original packets.
(2) In private test bed, attackers use the prepared attack packets to launch network attacks and gather network logs. Network logs contain packets from both normal users/servers and attackers, and are sorted in the time order.
(3) Attackers process collected raw data into data samples.
(4) Attackers feed data samples to the PCAW adversarial example generation tool to generate PCAW adversarial examples by only modifying changeable information in the attack packets.
(5) From PCAW adversarial examples, attackers create candidate stealthy attack packets.
(6) Attackers launch real-world network attacks by replaying known stealthy attack packets.

5.2. PCAW Adversarial Example Generation

One of the most important steps in the above flow is Step 4 - creating adversarial examples. To create adversarial examples from the original data samples, feature-based perturbations are usually used. Perturbation values are calculated based on the minimum component (feature) within the data sample, indicating the direction of how changes should be made to alter the model output. For example, if the perturbation for a feature is positive, increasing the value of this feature tends to change this data sample into an adversarial example. A problem that needs to be considered in this process is the existence of invariants, which are the packet features that should not be changed. To address this issue, we propose to use **mask** after the original perturbation is generated, as illustrated in Figure 4. Mask is defined per minimum component of one data sample. For ARP poisoning, every data sample is a sequence of 42 integers, so the mask is also a sequence of 42 integers. The masks determine which values are **changeable**, and which are not. A byte/bit with mask value 1 means it is changeable, and 0 means it is unchangeable. By applying this mask, perturbations values corresponding to those unchangeable features will be reset to 0, which means no changes will be applied to these features. In this way, when the adversarial examples are converted back to network packets, the invariants are not changed and the resulting network packets are still valid.

To find out which fields in the packets are changeable, we refer to the protocol specifications to rule out fields related to the packet integrity, such as checksum and length fields. In addition, though some values

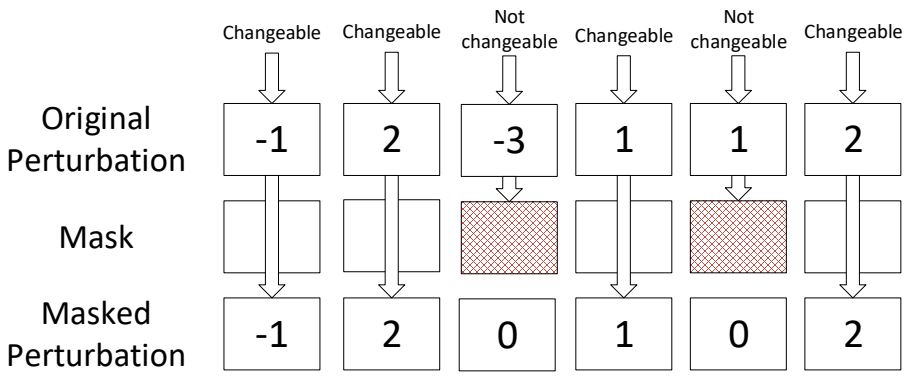


Fig. 4. Mask illustration.

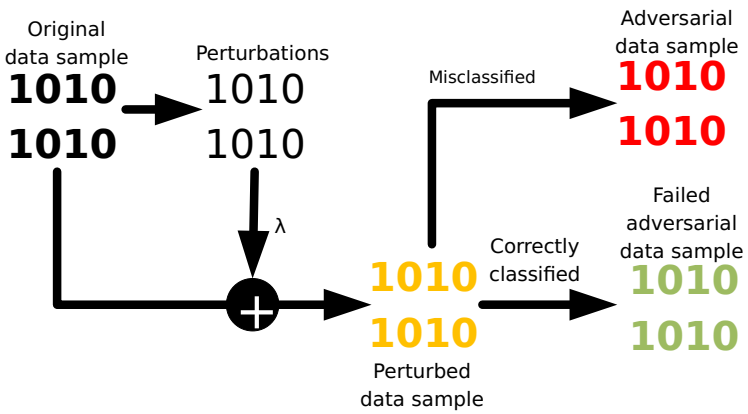
(such as certain reserved bits) are marked as unchangeable in the protocol specifications, the servers may just ignore them in practice, so it is still safe to change them during perturbation. We launched network attacks in our test bed to find out such fields. For example, one changeable field is the time-to-live (TTL) in the IP layer. Though commonly set to 64, programs usually do not check the TTL value and simply let packets pass regardless. Therefore, this field is changeable for attackers: no matter how this value is changed, this packet will still be accepted rather than abandoned.

After we have generated the mask, we apply it in two well-known adversarial example generation methods, namely FGSM and ZOO.

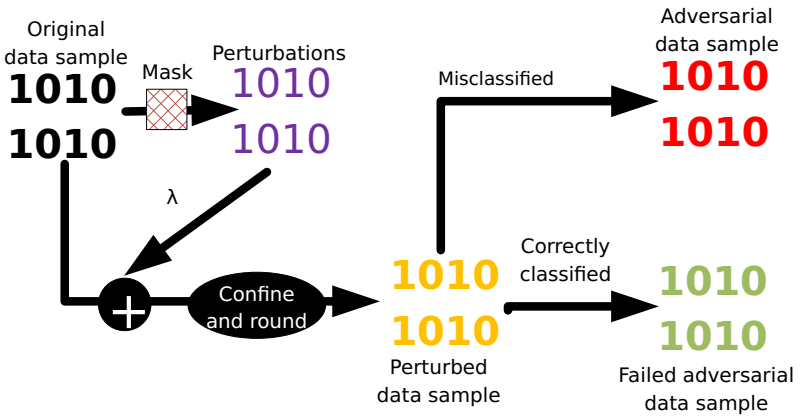
Proposed Gradient-based Attack. FGSM is a well-known gradient-based evasion attack towards neural networks. FGSM takes the inputs of one original data sample as the seed, and the original model to extract gradients. The general workflow is as illustrated in Figure 5a: 1) Given the original data sample and the target model, it calculates a perturbation specific to that provided seed with back propagation. 2) A perturbed data sample (X') is generated by adding the original data sample (X) and the product of the perturbation (J) and a magnitude parameter λ, denoted as $X' = X + \lambda * J$. 3) In the end, X' is validated using the target model. If X' is misclassified, then this X' is an adversarial data sample; if not, then this attempt fails. To find more adversarial data samples, attackers may change the magnitude parameter λ or the original/seed data sample X, so that more different X' could be tried.

We modify the original FGSM for application in network attacks, as illustrated in Figure 5b. Main changes include two parts: 1) masks are applied when generating perturbations; 2) during generating perturbed data samples, we confine and round all values to the valid range. Perturbations and the magnitude parameter may introduce decimal fraction or make values out of the valid range. For example, in ARP poisoning detection, every value should be integers within [0, 255]. In DNS cache poisoning, every value is either 0 or 1. Hence these values should be confined and rounded.

Proposed Score-based Attack. For score-based attack, we propose a method based on the Zeroth Order Optimization (ZOO) [26]. Similar to FGSM, ZOO attack generates perturbations specific to the input data sample. Instead of calculating the gradient on the actual target model via back propagation, the proposed score-based attack estimates the approximate gradient using a finite difference method using the probability score which is output by the target model. The untargeted ZOO attack uses a hinge-like loss function $f(x)$ to measure the gap between the confidence score $\mathbb{F}(x)$ of the original class label and the confidence scores of the other classes. It applies a small perturbation on original samples $x_{i1} = x + he_i$



(a) Original FGSM.



(b) Masked FGSM.

Fig. 5. Illustration for FGSM.

and $x_{i2} = x - he_i$, where e_i is a standard basis vector and h is a small constant. And then it estimates the gradient $g_i = \frac{\partial f(x)}{\partial x_i}$ using symmetric difference quotient. With one more objective function evaluation, the Hessian estimate h_i can be obtained. After estimating the gradient and Hessian for x_i , we can use any first or second order method to approximately find the best perturbation δ to evade the NN model.

The proposed score-based attack is illustrated in Algorithm 1. First, changeable fields are randomly selected from the mask and a standard basis vector e_i with only the i-th field as 1 is applied to obtain two symmetrical samples x_{i1} and x_{i2}. Then, those modified samples are fed to the target model $\mathbb{F}(\cdot)$ to retrieve predicted probabilities $\mathbb{F}(x_{i1})$ and $\mathbb{F}(x_{i2})$. Finally, Zeroth Order Stochastic Coordinate Descent with Coordinate-wise ADAM (ZOO-ADAM) is used to calculate the estimated gradient and the corresponding perturbation. The perturbation is not directly added to the original sample. The final perturbation is normalized and rounded to the valid range, which is $[0, 255]$ for ARP poisoning. Those steps repeat until the *niter* iterations are completed or the modified data sample successfully evades the target

Algorithm 1: Proposed score-based attack.

input : $\mathbb{F}(\cdot)$, which is the target model.
input : x, an original (seed) data sample.
input : *niter*, a control parameter to limit the max number of iterations.
input : $MASK$, the mask which indicates changeable fields.
output: $\widetilde{x}$
$i \leftarrow 0$;
while $i <$ *niter* **do**
  Randomly select a changeable field i from $MASK$;
  Apply symmetrical perturbations on the selected field $x_{i1} = x + e_i$ and $x_{i2} = x - e_i$;
  Retrieve the predicted probability $\mathbb{F}(x_{i1})$ and $\mathbb{F}(x_{i2})$;
  Calculate the loss function $f(x) = max\{log(\mathbb{F}(x))_{t0} - max_{i \neq t_0} log(\mathbb{F}(x))_i\}$;
  Estimate the gradient $g_i \approx \frac{f(x_{i1}) - f(x_{i2})}{2}$ and the Hessian estimate $h_i \approx \frac{f(x_{i1}) + f(x_{i2}) - 2f(x)}{h^2}$;
  Use ZOO-ADAM algorithm to calculate the perturbation δ;
  $\delta \leftarrow round(\frac{\delta}{\sum_i^n \delta})$;
  Update $x_i \leftarrow x_i + \delta$;
  Round x_i to the valid range;
  if *Successfully evade the target model* **then**
    return Adversarial sample $\widetilde{x} = x$;
  end
  $i \leftarrow i + 1$;
end

model.

5.3. Convert PCAW Adversarial Examples to Candidate Stealthy Attack Packets

As stated in Section 4, one of the reasons to choose ARP poisoning for demonstration is that, the conversion between data samples and network packets is straightforward. Basically, the back-conversion is a reverse process of data processing. In the case of ARP poisoning detection, each ARP packet is processed into one data sample with no information loss. Every byte in the original packet is preserved (except padding bytes) in the resulting data sample as an converted integer. Therefore, the back-conversion is to convert integers back to bytes in the original order to form a packet, and add padding bytes if needed. For more details, please refer to the data processing part in the work [6].

5.4. Mitigate Uncertainties for Multi-packet Attacks

In subsection 5.2, we ensure the packet's validity by using mechanisms such as masks, value confine and round, but these mechanisms do not ensure the session's validity. This subsection demonstrates how to use PCAW adversarial examples to launch stealthy multi-packet network attacks.

In Figure 6, we present the threat model for stealthy multi-packet network attacks using DNS cache poisoning as an example:

(1) Attackers prepare original attack packets.

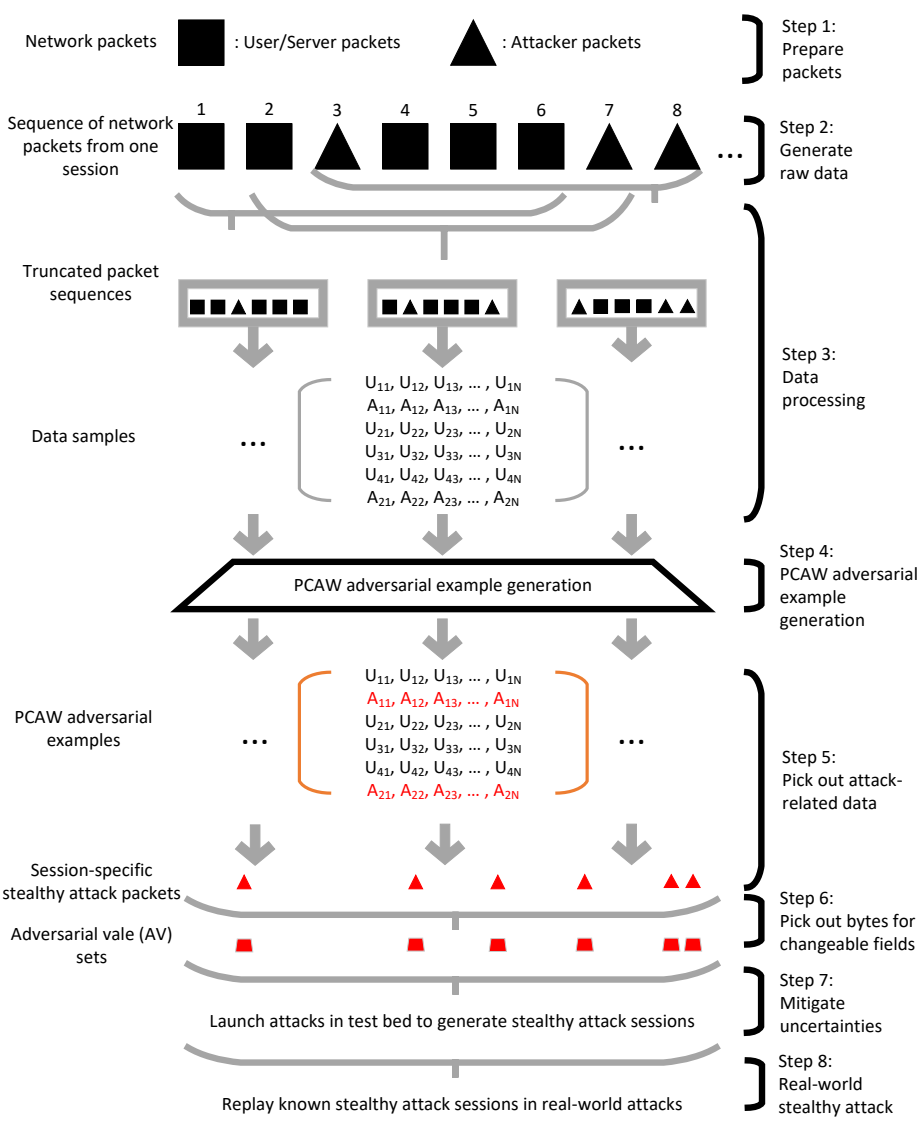


Fig. 6. The threat model for stealthy multi-packet network attacks.

(2) In their own test bed, attackers use the prepared attack packets to launch network attacks, and gather network logs at the same time. Network logs contain packets from both normal users/servers and attackers, and are sorted in the time order.
(3) Attackers truncate network logs into truncated packet sequences, which are in turn processed into data samples. Meanwhile, attack packets' orders in data samples are recorded, to be used later.
(4) Attackers feed data samples to the PCAW adversarial example generation tool to generate PCAW adversarial examples by only modifying changeable information in the attack packets, as the attack packets are the only ones that can be controlled by attackers.
(5) From PCAW adversarial examples, attackers pick out the bytes related to attack packets (using the

order information from before) if needed.

(6) From session-specific attack packets, attackers select bytes for changeable fields to form adversarial value (AV) sets.

(7) To mitigate the uncertainties discussed earlier, attackers use AV sets to simulate the network attacks in his/her own test bed. Any successful stealthy attack sessions will be recorded in the AV set database.

(8) Attackers launch real-world network attacks by replaying known stealthy attack sessions from his/her simulations.

In a DNS cache poisoning attack, packet 1 corresponds to DNS query sent by the user, packet 2 corresponds to DNS query sent by the local DNS server, packet 3 corresponds to attacker's spoofed DNS response, and packet 4 corresponds to global DNS server's DNS response. Packets 5 and 6 correspond to the remaining steps or other DNS packets, and packets 7 and 8 stand for dummy attack packets, the impact of which will be discussed in subsection 6.2. In the data samples, we use U to stand for bytes from user/server packets, and A to stand for bytes from attacker packets. The subscripts stand for position of bytes in the packets. For example, A_{12} stands for the second selected byte from the first attacker packet.

For step 5 and step 6, we extract values of the changeable fields from generated PCA adversarial examples to create multiple "adversarial values (AV) sets", each set corresponds to one original attack packet in one data sample. Step 6 and 7 are only needed for multi-packet attacks. Step 6 is necessary for DNS because the attack packets have to be crafted based on the user/server packets, and session-specific attack packets cannot be directly used in future attacks. That is why AV sets are extracted: they are expected to carry the adversarial effects to evade neural networks. Step 7 is to mitigate the uncertainties by generating stealthy attack sessions in attackers' own test bed. Those stealthy attack sessions can be used to build an AV set database, to which attackers can refer in later real-world attacks. With this database, a patient and cautious attacker can persist in the LAN while sniffing packets. Only when a perfect attack chance appears will the attacker take action, trying to replay the stealthy attack session the attacker already knows. In this way, the three problems raised earlier are bypassed.

To generate stealthy attack sessions, the following factors can be used to reduce detection rates (DR) and possibly generate more stealthy attack sessions.

Dummy Attacker Packets. For DNS cache poisoning, since the detection is based on several continuous network packets, a natural choice of attackers is to send packets more diligently so that more attacker packets will be included in data samples, and therefore these samples will more likely be misclassified as benign. We call these packets sent by attackers as **dummy attacker packets**. To maximize the chances of blending dummy attacker packets into data samples, these dummy packets should be as similar to the real packets as possible, because attackers have no information about the way in which the defenders will filter and process network logs.

Partial Orders. Partial order refers to the location of attacker packet(s) in data samples. It is another factor that is out of attackers' control. When network logs are processed into data samples, we divide them into different sessions and use a sliding window. All those processing is carried out at the defender's side. Therefore, attackers have no idea where those attacker packets are located in the data samples. The number of attacker packets that get into the data samples can be indirectly affected by utilizing dummy attacker packets, but the partial order of attacker packets in data samples cannot be controlled by attackers in any way. As a result, attackers can only get the expected detection rates with respect to the partial orders. Therefore, choosing a promising partial order distribution is important for attackers.

After attackers have got stealthy attack sessions in their own test bed, they can build a database structured as in Figure 7. The database consists of entry keys and sub-tables of key-value pairs. Each key

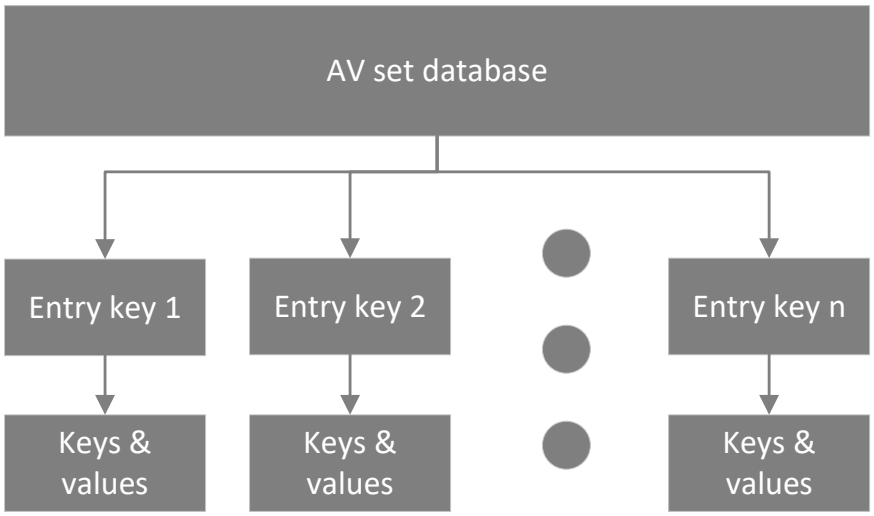


Fig. 7. AV set database structure.

is a hash value, each sub-table, marked by one specific entry key, represents a known stealthy attack session, and the values are the AV sets. Each hash value is calculated based on the selected bytes from previous (up to 5, because the windows size is 6) packets in the session. Hash values are used as keys instead of directly comparing raw data for the sake of performance, because DNS cache poisoning attack requires quick response at the attackers' side. If the entry key is matched, the corresponding stealthy attack session is selected to be replayed.

For example, in the DNS cache poisoning example, one session contains eight packets $S = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$, of which x3 and x7 are attacker packets. Let's assume that this session is a stealthy attack session. The entry key is calculated with $[x_1, x_2]$, and the sub-table following this entry key has two rows: key $hash([x_1, x_2])$ corresponding to x_3, and key $hash([x_2, x_3, x_4, x_5, x_6])$ corresponding to x_7.

After the AV set database is built, we use it to simulate a patient and cautious attacker. Such attacker will always wait for a perfect chance to take action. That is, this attacker will only send out attacker packets when the observed packets' hash has a perfect match in the database. Even if the entry key is matched, as long as the observed packets at a later time does not match hashes in the sub-table, the attacker stops at once and let go of the current session. Taking the earlier example, the attacker will first wait for $[x_1, x_2]$, and then send out x_3. Only if this session is followed by $[x_4, x_5, x_6]$, the attacker will send out x_7, completing the replay of a known stealthy attack session. If not, the attacker stops at once and gets back to sniff for $[x_1, x_2]$.

AV Set Selection. Subsection 5.1 mentioned that the the changeable fields of attacker packets will be replaced with a chosen AV set to generate the stealthy attack packets. The selection of AV set may affect the detection rate of neural networks. The attack packet is a direct response to the local DNS server's query packet, so the selection of AV set in the stealthy attack packet should be based on that query packet.

Therefore, we propose to build a dictionary that can be used to effectively search for the proper AV sets. The dictionary consists of multiple entries; and each entry contains a key and one or more values. The keys in the dictionary are the DNS query packets sent out by the local DNS server. The values are the AV sets. This dictionary can be created in the following way:

(1) Map local DNS server's query packet to the attack packet. This map is one-to-one.
(2) Map the attack packet to the AV set. This map is one-to-one or one-to-none, because not all data samples can be used to generate PCAW adversarial examples.
(3) With the two steps above, establish the mapping between local DNS server's query packet (as key) and the AV set (as value).
(4) Merge key-value pairs that have the same key.

When use the dictionary, we need to choose the AV sets based on the keys. However, in actual attacks, we cannot guarantee that there is always an exactly matching key for the local DNS server's query packet. Therefore, the distances between the query packet and the keys can be measured to find out the closest key. Depending on how the distance is defined, one observed local DNS query packet may refer to different entries. If the chosen entry holds multiple AV sets (because of merging key-value pairs), one of them is randomly chosen. Assuming x_i represents a byte in the local DNS server's query packet, and x'_i represents a byte at the same position in the key, there are four commonly used distance measuring methods:

- $L_0 = count(x_i \neq x'_i)$.
- $L_1 = \sum_i |x_i - x'_i|$.
- $L_2 = \sqrt{\sum_i (x_i - x'_i)^2}$.
- $L_{inf} = \max(|x_i - x'_i|)$.

With the distance measuring method, the closest key is selected for each observed local DNS server query packet, and the corresponding entry is also determined. Please note that such distance measuring introduces additional processing time at the attacker's side. However, for DNS cache poisoning to succeed, attacker's packet has to arrive earlier than the global DNS server's packet. To rule out the impact of the additional distance measuring time on the attack success, in our experiments, we manually insert time lags between the local DNS server and the global DNS server's communications, so that attack packets always arrive earlier.

6. Evaluations

In this section, we will evaluate the effectiveness and efficiency of our proposed methods. Specifically, we want to answer the following evaluation questions:

- Q1: How effective are our proposed stealthy network attack against ARP poisoning detection?
- Q2: How effective are our proposed stealthy network attack against DNS cache poisoning detection?
- Q3: How costly are our proposed methods?

All experiments are performed using a Windows machine equipped with an Intel Core i9-9900KS CPU and Nvidia RTX 3090 for accelerating computations. For software, we use Python 3.8.5, TensorFlow 2.4.1 with GPU support, ART 1.6.1 [34], and FoolBox 3.3.1 [35, 36]. All data samples and trained detection models are retrieved from another work [6].

Table 4
Proposed PCAW adversarial example generation methods' performances towards ARP poisoning detection.

Proposed methods	Time (s)	PCAWAE [1]	Memory (MB)	Suc. ARP Atk [3]	Average losses			
					L_0	L_1	L_2	L_{inf}
Masked Random [2]	154.03	0	714.35	0	NA			
Masked FGSM	0.03	0	687.22	0				
Masked ZOO	602.90	36	33.57	23	11	1214.17	418.43	210.67

[1] PCAWAE: Number of PCAW adversarial examples generated.
[2] Masked random: Randomly changes the changeable portions regardless of gradients. This serves as a baseline.
[3] Suc. ARP Atk: Number of successful stealthy ARP poisoning attacks.

6.1. Q1: Stealthy ARP Poisoning

Phase 1: PCAW adversarial example generation. We studied the performance of different PCAW adversarial example generation methods for ARP poisoning attack, including masked FGSM and masked ZOO. Table 4 shows results. We also added a category of "masked random" as the baseline, in which we only randomly changes the changeable portions regardless of gradients. For each method tried, we randomly select 100 data samples as the seeds to feed into PCAW adversarial example generation. The results show that masked FGSM is not effective, but is very fast; on the contrary, masked ZOO is slow, but generates more PCAW adversarial examples. The reason that masked FGSM fails to generate any PCAWAE is because we implement them in a different way. Masked FGSM changes multiple features in the data samples at a time, but masked ZOO changes one feature at a time. Therefore, masked ZOO works on a smaller granularity and can potentially find more PCAWAE. As a result, in Table 4, though masked ZOO finds 36 PCAWAEs, masked FGSM finds none.

Phase 2: Launching stealthy ARP poisoning attacks. As shown in Table 4, masked FGSM produces 0 PCAW adversarial examples, which corresponds to 0 candidate stealthy attack packet, while masked ZOO produces 36 PCAW adversarial examples and 36 corresponding candidate stealthy attack packets. The 36 candidate stealthy attack packets are then used to launch 36 ARP poisoning attacks. Results show that 23 of those are actual stealthy attack packets that can lead to successful stealthy attacks.

We inspect the remaining 13 failed candidate stealthy attack packets that lead to failed ARP poisoning attacks, and find out that the IP addresses in those packets are modified to an invalid address, such as 192.168.100.0. This is because when the mask for ARP poisoning is created, the whole last byte (corresponds to the "0" in the IP) in the IP address field is marked as changeable. The masked ZOO algorithm can thus change it to any integer in [0, 255]. However, addresses like 192.168.100.0 are usually reserved for broadcasting, and are not mapped to any particular machine in this LAN. That is why ARP poisoning attack attempts using such IP addresses fail.

Since 23 of the candidate stealthy attack packets are actual stealthy attack packets, the overall stealthy attack success rate of the masked ZOO approach is $23/36 \approx 63.8\%$.

6.2. Q2: Stealthy DNS Cache Poisoning

Phase 1: PCAW adversarial example generation.

As discussed earlier, more attack packets in the data sample give attackers more chances to modify the data sample, and thus more PCAW adversarial examples can likely be generated. Therefore, we inspect all data samples to find out how many attack packets each data sample contains. The majority of all data samples contains only one attack packets, so we randomly choose 100 of them to generate the PCAW

Table 5
Proposed PCAW adversarial example generation methods' performances towards DNS cache poisoning detection.

Proposed methods	Data samples representing one attack packet[1]				Data samples representing two attack packets[1]			
	Time (s)	Seed [2]	PCAWAE [3]	Memory [5]	Time (s)	Seed[2]	PCAWAE [3]	Memory [5]
Masked random [4]	234.47	100	0	2.41	231.39	89	0	2.15
Masked FGSM	0.03	100	0	138.96	0.16	89	3	41.52
Masked ZOO	2148.92	100	0	39.22	1768.66	89	5	26.02

[1] There are no data samples representing three or more attack packets.
[2] Seed: Number of original data samples fed to the generation methods.
[3] PCAWAE: PCAW adversarial examples.
[4] Masked random: Randomly changes the changeable portions regardless of gradients. This serves as a baseline.
[5] Memory: The memory taken in MB.

adversarial examples. For the other 89 data samples (less than 1% of all data samples) that contain two attack packets, we use all of them. The PCAW adversarial example generation results are presented in Table 5. It is generally more difficult to generate PCAW adversarial examples for DNS cache poisoning than for ARP poisoning. This is not surprising because: 1) The DNS cache poisoning detection neural network is more accurate than that of ARP poisoning, as shown in Section 2; 2) The ratio of changeable portions in data samples is smaller than that of ARP poisoning. The attack packets are only 1/6 or 1/3 of an data sample, and only part of the attack packet are changeable. Despite of the difficulties, our proposed methods still succeed in generating PCAW adversarial examples, as shown in Table 5. The table also shows that, if there are two attack packets in the data sample, PCAW adversarial example generation is easier than if there is only one.

Phase 2: Uncertainties mitigation.

After PCAW adversarial examples are generated, attackers extract AV sets and use them to launch DNS cache poisoning attacks in the test bed in order to generate stealthy attack sessions which the attackers can replay in real-world attacks. There are several factors that can help attackers to decrease DR on the data sample level and lead to more stealthy attack sessions: dummy attack packets, AV set selection, and partial orders.

Impact of Dummy Attack Packets. To evaluate the impact of dummy attack packets, the attacker machine not only sends out attack packets as usual when an attack chance appears, but also sends out dummy attack packets periodically, regardless of whether there is an attack chance or not.

The results are shown in Table 6. Our observation is that the DRs of neural networks decrease as the numbers of dummy packets in a data sample increase. In another word, more dummy packets lead to lower DRs. When there is no dummy attack packets, or dummy attack packets are sent less frequently than every 0.3 second, the average DRs on the data sample level are above 90%. (There are less than 2 attack packets in every data sample, on average. The less frequently dummy attack packets are sent, the less attack packets get into data samples.) When the time interval drops to 0.2 second, DR drops to about 78%. (There are more than 2 attack packets in every data sample, on average.) When the time interval further drops to 0.1 second, DR drastically drops to about 55%. (There are more than 3 attack packets in every data sample, on average.) However, a careful examination of all the DNS cache poisoning attack sessions reveals that out of the 33177 attack sessions in total, only 2 of them are stealthy attack sessions, as shown in Table 7. This means, though dummy attack packets help lower DRs, using it alone may not be efficient enough for attackers to generate stealthy attack sessions.

Detection Rates with respect to Partial Orders. We have conducted statistics and find that detection rates corresponding to different partial orders are very polarized. Full results are shown in Figure 8. The

Table 6
DNS cache poisoning with/without dummy attacker packets.

Dummy packet interval[1](s)	None	1	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Total [2]	10745	13846	14609	14792	15462	16126	17068	18202	20941	25120	36150
Mis. [3]	66	176	232	251	268	274	421	671	1472	5489	16181
DR [4]	99.39%	98.73%	98.41%	98.30%	98.27%	98.30%	97.53%	96.31%	92.97%	78.15%	55.24%
Rep. [5]	1.06	1.25	1.26	1.30	1.35	1.40	1.49	1.60	1.79	2.26	3.36

[1] Dummy interval: The time interval to send out dummy attacker packets. "None" means no dummy attacker packets are sent out.
[2] Total number of data samples.
[3] Number of misclassified data samples.
[4] DR: Detection rate.
[5] Average number of attack packets represented in the data samples.

Table 7
Statistics and detection results with respect to dummy interval on session level.

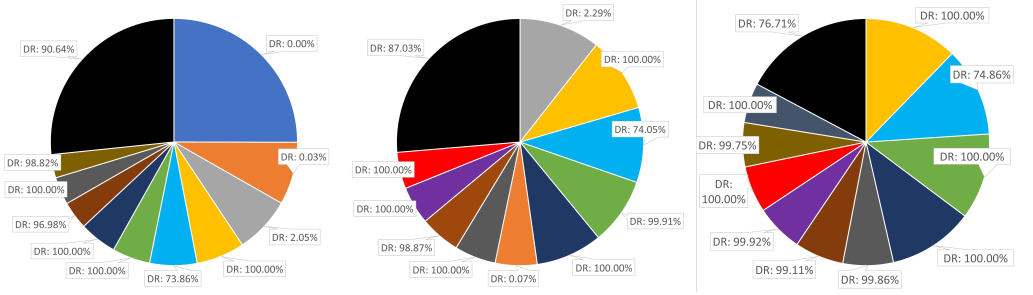
	Data samples per session*			Sessions	
	Min	Max	Avg	Total	Detected
No dummy	1	16	4.04	2872	2872
Dummy 1s	1	15	4.83	3051	3051
Dummy 0.9s	1	46	5.16	3021	3021
Dummy 0.8s	1	211	5.27	3011	3011
Dummy 0.7s	1	221	5.37	3060	3060
Dummy 0.6s	1	16	5.58	3025	3025
Dummy 0.5s	1	68	6.02	3022	3021
Dummy 0.4s	2	218	6.32	3021	3021
Dummy 0.3s	2	19	7.07	3049	3048
Dummy 0.2s	3	30	8.56	3019	3019
Dummy 0.1s	1	26	12.19	3026	3026

* The number of data samples in sessions.

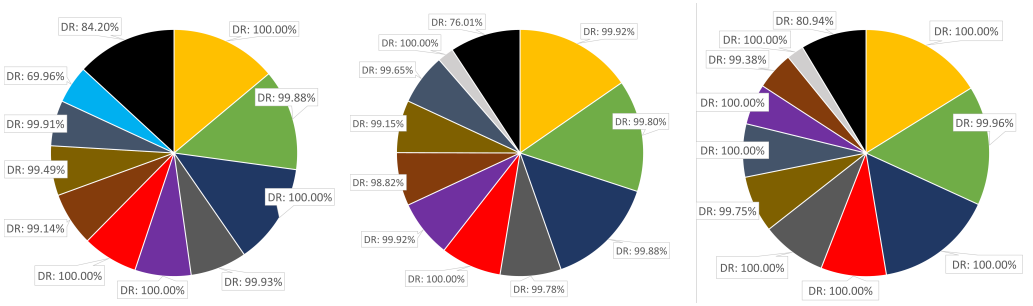
majority of detection rates are above 90%; some are about 70%; and others are below 5%. None of them are between 5% to 65%.

Generally speaking, less frequent dummy attack packets result in the high detection rates. For example, when dummy packets are sent out every 0.1s, one-third (of 3 partial orders or pies) of the data samples have DRs close to 0%; when the time interval is 0.2s, the rate drops to about 10% (of just 1 partial order or pie); when the time interval becomes greater than 0.2s, the most common ten partial orders all have high detection rate. In a word, the distribution of partial orders is not a uniform distribution, different partial orders have different DRs, and how frequently dummy attack packets are sent have a big influence on partial order distribution.

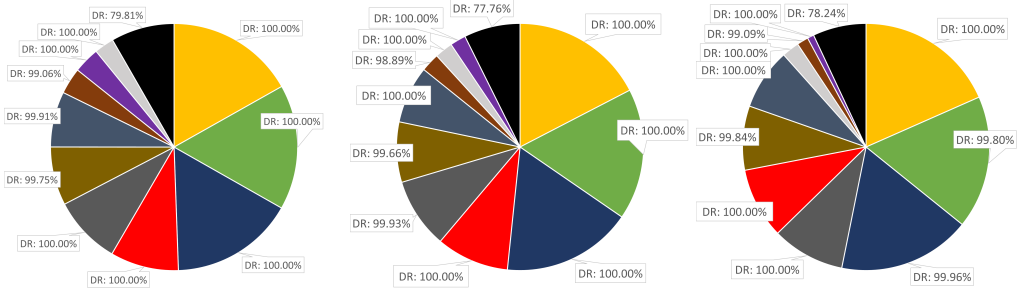
Impact of AV set Selection. To build the dictionary that helps to select the suitable AV set, we have conducted additional DNS cache poisoning attacks to collect data and generate PCAW adversarial examples with masked FGSM (Masked ZOO is too time-consuming so it is not used here.) After launching the DNS cache poisoning attack for 60000 times with dummy attack packets, a total of 244340 data samples are generated, which result in a dictionary with 148 keys holding 202 AV sets.



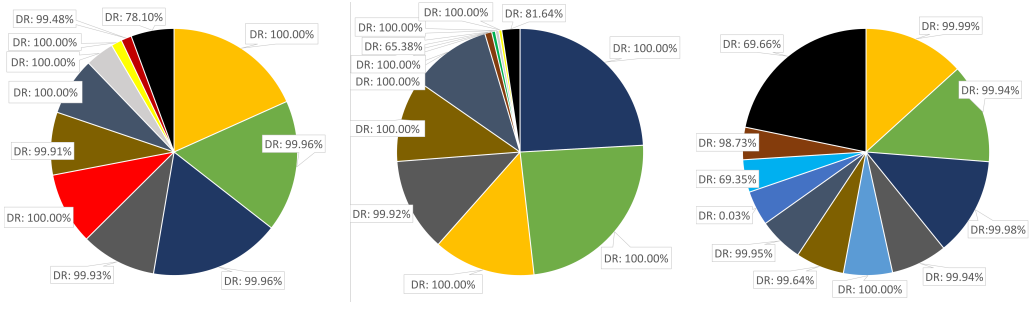
(a) Dummy packets every 0.1s. (b) Dummy packets every 0.2s. (c) Dummy packets every 0.3s.



(d) Dummy packets every 0.4s. (e) Dummy packets every 0.5s. (f) Dummy packets every 0.6s.



(g) Dummy packets every 0.7s. (h) Dummy packets every 0.8s. (i) Dummy packets every 0.9s.



(j) Dummy packets every 1s. (k) No dummy packets. (l) Overall distribution.

Fig. 8. Partial order distributions and corresponding detection rates.

Table 8
Detection rates with respect to the number of different distance measuring methods.

Type	Number of attack packets represented in each data samples						Average
	1	2	3	4	5	6	
	DR	DR	DR	DR	DR	DR	DR
Random [1]	99.99%	100.00%	-				99.99%
Random [1]with dummy [2]	99.72%	98.87%	90.33%	39.51%	18.72%	8.80%	79.84%
L_0	99.99%	100.00%	-				99.99%
L_0 with dummy [2]	99.73%	95.58%	58.37%	4.73%	0.24%	0.28%	66.02%
L_1	100.00%	100.00%	-				100.00%
L_1 with dummy [2]	99.77%	98.73%	82.54%	46.63%	39.23%	26.27%	81.30%
L_2	100.00%	100.00%	-				100.00%
L_2 with dummy [2]	99.80%	98.91%	81.77%	47.50%	38.16%	28.80%	80.69%
L_{inf}	100.00%	100.00%	-				100.00%
L_{inf} with dummy [2]	99.78%	98.13%	72.19%	9.63%	0.12%	0.00%	69.12%

[1] "Random" means the dictionary is not used and AV sets are selected totally randomly. The two "random" cases serve as the baselines.
[2] Dummy attack packets, if used, are sent out every 0.1s.

We tried all the distance measuring methods mentioned in subsection 5.4, and experiment results are summarized in Table 8. Each row shows results for one method (with 5000 attack sessions conducted for each method respectively), and each column shows the detection rates with respect to the number of attack packets represented in one data sample. The last column shows the average numbers of represented attack packets and the average detection rates. Similar to Table 6, the more attack packets represented in one data sample, the less likely the attack can be detected. When no dummy attack packets are sent out, DRs are almost 100%. When dummy attack packets are used, 1) L_1 and L_2 distance measuring methods fail to decrease detection rates, while L_0 and L_{inf} succeed in decreasing detection rates; 2) L_0 distance measuring achieves the lowest detection rates. Comparing to "random with dummy", the detection rate drops from 79.84% to 66.02%.

In addition to the data sample level, we have also inspected the collected data on the session level. To our surprise, the results show that none of the sessions in the ten cases shown in Table 8 are stealthy attack sessions, which probably means they can all be detected. This indicates that though AV set selection can effectively lower the DRs at data sample level on general, it is not effective at the session level. A closer look at the sessions and the data samples also shows that, it is usually the first several data samples in the session that cannot get misclassified. Such data samples have only 1 or 2 attack packets inside. The reason is that the local DNS server, on receiving the user machine's query packet, will send out several packets successively: query the IP address of the desired domain name, query the authoritative DNS server, query the IP addresses of root DNS servers, etc. As a result, every session is certain to have data samples that have very few attack packets, and the DR on the session level cannot be effectively lowered.

Phase 3: Launching stealthy DNS cache poisoning attacks.

We build an AV set database with the two stealthy attack sessions we found, and use it to simulate a patient and cautious attacker. If the attacker takes no action in a DNS session, it is referred as a "no-attack session"; if the attacker takes some action(s), but aborts before the attack is finished and does not fully replay the template session, it is referred as an "aborted session"; if the attacker fully replays the template session, it is referred as a "replayed session". In our experiment, it takes about four days with 46545 DNS sessions, to get a perfect attack chance. The 46545 DNS sessions contains 46541 no-

attack sessions, 3 aborted sessions, and the last one replayed session. We have also verified that none of the no-attack sessions are reported as false positive; all of the aborted sessions are correctly detected as malicious (stealthy attack session is not fully replayed, so aborted sessions cannot guarantee to be stealthy); for the one replayed session, it is not detected, and the attack succeeds.

Though it seems that a perfect chance is hard to wait, considering that attackers can even wait for months to take actions in APT campaigns, we believe several days is realistic and acceptable for attackers. To provide an estimation of the waiting time, we have the following observation: 1) in a large-scale enterprise network, a typical local DNS server's network throughput is about 6 Mbps, namely $6 * 1024 * 1024/8$ bytes per second. 2) Half of all the local DNS server's network traffic is DNS traffic. 3) An average DNS packet is 170 bytes in length. 4) A DNS session, if the entry is already cached (no attack chance), can be as short as 2 packets; if the entry is not cached (potential attack chance) and attackers are not involved, the session can contain at least 4 packets; or, if the attacker launched the DNS cache poisoning attack, the session can be even longer (not to mention if dummy attack packets are used). In the worst case, we assume that every DNS session, on average, contains 12 packets, which means $6*1024*1024/8/2/170/12 \approx 193$ sessions per second. Therefore, even if there are as few as one potential attack chance out of 1000 DNS sessions, every $1000/386 \approx 5.181$ seconds, there will be one attack chance. Considering the results in the previous paragraph, every $5.181 seconds * 46545 \approx 67 hours$, attackers can expect one successful stealthy DNS cache poisoning attack. Also, our results are achieved with a database containing two stealthy attack sessions. If the attacker invests more time and efforts for preparations, and builds the database with more stealthy attack sessions, the estimated waiting time can be further shortened.

6.3. Q3: Costs of the Proposed Methods

Time and Memory Consumption. When generating PCAW adversarial examples, we also monitor the time and memory consumption. Memory consumption is measured by computing the memory usage increase caused by the PCAW adversarial example generation. From Table 4, it can be observed that masked FGSM needs exceptionally less time, but tends to have more memory consumption than masked ZOO. Masked random uses the most time and memory, because it is designed to loop until an adversarial example is found, or the algorithm reaches the maximum iterations. Compared to FGSM and ZOO, the random method does not have any guideline to generate perturbations, so it becomes the most inefficient approach.

PCAW Adversarial Examples' Losses. In the area of adversarial learning, losses measure the distance of adversarial examples to their seed data samples, which indicate to what extent a data sample has to be modified so that it becomes an adversarial example. Assuming x_{ij} represents a feature in the seed data sample, and x'_{ij} represents a feature in the adversarial example, we use the similar distance measuring methods L_0, L_1, L_2, and L_{inf} mentioned earlier. We have measured the average losses of our generated PCAW adversarial examples. For ARP poisoning, no PCAW adversarial examples are generated with masked FGSM, so losses are only calculated for those generated by masked ZOO. The L_0 average loss is 11, L_1 is 1214.17, L_2 is 418.43, and L_{inf} is 210.67. It means that about 11 bytes (L_0) in the ARP packets need to be modified, and the values (L_{inf}) may be changed by 210.67. For DNS cache poisoning, depending on how many attack packets are represented in the data sample, how large the changeable portion varies, so they are listed separately. If there is only one attack packet represented, no PCAW adversarial examples are generated, so losses cannot be calculated. If there is two attack packets represented, masked FGSM can generate PCAW adversarial examples with relatively less modifications.

Table 9
PCAW adversarial examples' average losses.

Attack	Measurements	Masked FGSM	Masked ZOO
ARP	L_0	- *	11
	L_1		1214.17
	L_2		418.43
	L_{inf}		210.67
DNS (one attack packet in data sample)	L_0	- *	
	L_1		
	L_2		
	L_{inf}		
DNS (two attack packets in data sample)	L_0	18.33	23.83
	L_1	18.33	23.83
	L_2	4.26	4.87
	L_{inf}	1	1

* Some results cannot be presented because there are no PCAW adversarial examples in those cases, and losses cannot be calculated.

The L_0 average loss is 18.33, L_1 is 18.33, L_2 is 4.26, and L_{inf} is 1. Because every feature for data samples of DNS poisoning detection is either 0 or 1, $L_0 == L_1$, and L_{inf} can only be 0 or 1.

7. Discussion

7.1. Limitations

Attack success rates and stealthiness are usually the top concerns of attackers, especially for the ones conducting Advanced Persistent Threats (APTs). We have proposed new evasion methods to generate malicious adversarial examples, so that attack success rate and stealthiness are both preserved. However, as shown in section 6, they are not very efficient. For network attacks that need multiple network packets to launch and detect, such as DNS cache poisoning, sending dummy attacker packets is an effective way to substantially decrease the chances of being detected by neural networks. Nevertheless, dummy packets can also increase the attackers' risks of being noticed due to the abnormal packet contents and the increased interaction with the target machines or servers.

Therefore, it is very challenging for attackers to find a perfect solution that can preserve network attack success rate, stealthiness, and efficiency all at once. Attackers will need to make trade-offs among these aspects. As for defenders, it is recommended to combine multiple detection approaches rather than just using one, such as outlier detectors.

7.2. Counter Measures

The previous sections presented a novel attack type with PCAWAEs. This attack is stealthier than traditional network attacks and can still result in the malicious impact. What is more, because attackers modify the contents of the packets, network flow-based IDS may not be effective detecting this attack. However, because the PCAWAE generation is still based on observed adversarial example generation methods, PCAWAE may be rejected if adversarial learning techniques are applied.

This paper focuses on the attacking aspect, so we will only briefly discuss the counter measures. Generally speaking, adversarial learning techniques can be categorized based on the time point of deployment [37]: before or after the original model is produced. Some common adversarial learning techniques include adversarial training [7], where adversarial examples are augmented into the training data to train a robust model, defensive distillation [38], where a separate model is learned based on the output of a previous model, and data compression [39], where the original data samples are compressed to mitigate the effect of attacker perturbations. However, adversarial training needs a lot of adversarial examples. In our case, they should be PCAWAEs, which are not easy to generate. Defensive distillation does not need adversarial examples as input, but it is not effective against some advanced attacks [23, 40]. Unlike images, data compression may break some intrinsic patterns within network packet data. Therefore, instead of the above-mentioned common techniques, we select DeepCloak [41], which adds a mask layer just before the output layer of the detection model, so that extracted features that are affected by adversarial examples most will be filtered out.

We applied DeepCloak to the ARP poisoning detection neural network. As shown in Figure 1, the output layer contains 2 units for binary classification, and there are 35 extracted features from the previous layer. Our experiment results show that, when 1 extracted feature is filtered out, out of the 36 PCAWAEs generated by masked ZOO, 33 of them are now correctly predicted as malicious, and changes to the evaluation metrics (accuracy, recall, precision, and F1 score) on the original dataset is about 0.0001. We continue to increase the number of filtered out extracted features, and find out that the maximum number of reverted PCAWAEs is 34. When the number of filtered out extracted features is nearing half of all extracted features, the model's evaluation metrics will decrease drastically. In conclusion, though DeepCloak may not be able to revert all PCAWAEs' predicted labels to the correct labels, it is still an effective defense approach because it can revert about 90% of the PCAWAEs' predicted labels.

8. Related Work

In the domain of network security, all the existing work on generating adversarial examples are under the data manipulation threat model. In particular, we break them down into two groups.

Packet content manipulation. In [9] and [10], the authors show how a NN-based detector which works on network flow data can be evaded. In [9], the original malicious network traffic is mutated to transfer its features to the closest adversarial ones, and the authors used a special distance metric between two feature vectors to help solve the corresponding bi-level optimization problem. However, because no network/protocol constraint is considered in solving the optimization problem, the generated adversarial examples are *not* protocol-constraint-aware.

In [10], the authors conducted systematic experiments to evaluate whether the generated adversarial examples can evade the NN-based detectors. However, since the data manipulation threat model is assumed in this study, the authors did not propose any method for using the generated adversarial examples to generate the adversarial attack packets. Without these adversarial attack packets, the authors did not launch any real network attacks in their experiments. Another main difference between [10] and our work is as follows: the network constraints considered in [9, 10] focus on transport layer protocols (e.g., TCP and UDP); in contrast, because we aim to launch real network attacks under the standard threat model, the protocol constraints considered in our work are specific to individual attacks. For example, for the DNS cache poisoning attack, we focus on the constraints specific to the DNS protocol.

In [42], the authors evaluated the effectiveness of adversarial attacks against Botnet detectors. However, the adversarial attacks used in this study randomly modify the values of up to 4 different features:

flow duration, sent bytes, received bytes, and exchanged packets. Since the values are randomly modified, the generated adversarial examples are *not* protocol-constraint-aware.

Another work [43] proposes to modify packets themselves, but it only pads the packets with redundant bits, making the packets in question easy to be filtered out and noticed by security operators.

Timing manipulation and packet injection. The work conducted in [44] assumes that attackers can split large Transmission Control Protocol (TCP) data (to be sent) into multiple smaller packets, and generates adversarial examples by maliciously (a) manipulating the timing for sending out packets and (b) injecting fake packets. Note that our approach neither manipulates timing information, nor injects any fake packets.

9. Conclusion

With deep-learning-based detection systems being increasingly deployed in enterprise networks, if the adversary continues to launch network network attacks (as usual) without any adjustment, their APT campaigns would become significantly less stealthy. Noticing that none of the existing adversarial example generation methods could generate malicious packets that can simultaneously compromise the target machine and evade the neural network detection model, we introduced a practical way for launching stealthy network attacks, and proposed the approach of implementing this type of attack. We used ARP poisoning and DNS cache poisoning attacks as the case study to demonstrate the effectiveness of the proposed method.

References

[1] Snort - Network Intrusion Detection & Prevention System, 2021, [Online; accessed 15. Jun. 2021]. https://www.snort.org.
[2] The Zeek Network Security Monitor, 2023, [Online; accessed 21. Feb. 2023]. https://zeek.org.
[3] Y. Mirsky, T. Doitshman, Y. Elovici and A. Shabtai, Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection (2018), arXiv:1802.09089 [cs]. doi:10.48550/arXiv.1802.09089. http://arxiv.org/abs/1802.09089.
[4] Y. Zhang, X. Chen, D. Guo, M. Song, Y. Teng and X. Wang, PCCN: Parallel Cross Convolutional Neural Network for Abnormal Network Traffic Flows Detection in Multi-class imbalanced Network Traffic Flows, *IEEE Access* (2019), 1–1. https://ieeexplore.ieee.org/abstract/document/8787567/.
[5] M.D. Hossain, H. Ochiai, D. Fall and Y. Kadobayashi, LSTM-based Network Attack Detection: Performance Comparison by Hyper-parameter Values Tuning, in: *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, IEEE, 2020, pp. 62–69.
[6] Q. Zou, A. Singhal, X. Sun and P. Liu, Deep learning for detecting logic-flaw-exploiting network attacks: An end-to-end approach, *Journal of Computer Security* (2021), 1–30.
[7] I.J. Goodfellow, J. Shlens and C. Szegedy, Explaining and harnessing adversarial examples, *arXiv preprint arXiv:1412.6572* (2014).
[8] X. Yuan, P. He, Q. Zhu and X. Li, Adversarial examples: Attacks and defenses for deep learning, *IEEE transactions on neural networks and learning systems* **30**(9) (2019), 2805–2824.
[9] D. Han, Z. Wang, Y. Zhong, W. Chen, J. Yang, S. Lu, X. Shi and X. Yin, Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors, *IEEE Journal on Selected Areas in Communications* **39**(8) (2021), 2632–2647.
[10] R. Sheatsley, N. Papernot, M.J. Weisman, G. Verma and P. McDaniel, Adversarial examples for network intrusion detection systems, *Journal of Computer Security* **30**(5) (2022), 727–752. doi:10.3233/JCS-210094.
[11] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z.B. Celik and A. Swami, The limitations of deep learning in adversarial settings, in: *Proc. IEEE European Symposium on Security and Privacy*, 2016.
[12] Home - Suricata, 2021, [Online; accessed 15. Jun. 2021]. https://suricata.io.
[13] infosecdr, spade, 2021, [Online; accessed 15. Jun. 2021]. https://github.com/infosecdr/spade.

[14] X. Yuan, C. Li and X. Li, DeepDefense: Identifying DDoS Attack via Deep Learning, in: *2017 IEEE International Conference on Smart Computing, SMARTCOMP 2017*, 2017. https://ieeexplore.ieee.org/abstract/document/7946998/.
[15] Splunk and Tensorflow for Security: Catching the Fraudster with Behavior Biometrics, 2017, [Online; accessed 15. Jun. 2021]. https://www.splunk.com/en_us/blog/security/deep-learning-with-splunk-and-tensorflow-for-security-catching-the-fraudster-in-neural-networks-with-behavioral-biometrics.html.
[16] Training Transformers for Cyber Security Tasks: A Case Study on, 2021, [Online; accessed 15. Jun. 2021]. https://www.fireeye.com/blog/threat-research/2021/01/training-transformers-for-cyber-security-tasks-malicious-url-prediction.html.
[17] Fortinet Introduces Self-Learning Artificial Intelligence Appliance for Sub-Second Threat Detection, 2021, [Online; accessed 15. Jun. 2021]. https://www.fortinet.com/corporate/about-us/newsroom/press-releases/2020/introduces-self-learning-artificial-intelligence-appliance-for-sub-2nd-threat-detection.
[18] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi and C.-J. Hsieh, EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples, *Proceedings of the AAAI Conference on Artificial Intelligence* **32**(11) (2018). https://ojs.aaai.org/index.php/AAAI/article/view/11302.
[19] A. Kurakin, I. Goodfellow and S. Bengio, Adversarial examples in the physical world, *arXiv:1607.02533 [cs, stat]* (2017), arXiv: 1607.02533. http://arxiv.org/abs/1607.02533.
[20] S.-M. Moosavi-Dezfooli, A. Fawzi and P. Frossard, Deepfool: a simple and accurate method to fool deep neural networks, in: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.
[21] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z.B. Celik and A. Swami, The Limitations of Deep Learning in Adversarial Settings, in: *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, 2016, pp. 372–387–. doi:10.1109/EuroSP.2016.36.
[22] T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae and S. Ishii, Distributional Smoothing with Virtual Adversarial Training, *arXiv:1507.00677 [cs, stat]* (2016), arXiv: 1507.00677. http://arxiv.org/abs/1507.00677.
[23] N. Carlini and D. Wagner, Towards Evaluating the Robustness of Neural Networks, in: *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 39–57. ISSN 2375-1207. doi:10.1109/SP.2017.49.
[24] H. Hosseini, B. Xiao, M. Jaiswal and R. Poovendran, On the Limitation of Convolutional Neural Networks in Recognizing Negative Images, in: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2017, pp. 352–358–. doi:10.1109/ICMLA.2017.0-136.
[25] U. Jang, X. Wu and S. Jha, Objective Metrics and Gradient Descent Algorithms for Adversarial Examples in Machine Learning, in: *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, Association for Computing Machinery, 2017, pp. 262–277–. ISBN 978-1-4503-5345-8. doi:10.1145/3134600.3134635.
[26] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi and C.-J. Hsieh, ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models, in: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, Association for Computing Machinery, 2017, pp. 15–26–. ISBN 978-1-4503-5202-4. doi:10.1145/3128572.3140448.
[27] W. Brendel, J. Rauber and M. Bethge, Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models, *arXiv:1712.04248 [cs, stat]* (2018), arXiv: 1712.04248. http://arxiv.org/abs/1712.04248.
[28] A. Madry, A. Makelov, L. Schmidt, D. Tsipras and A. Vladu, Towards deep learning models resistant to adversarial attacks, *arXiv preprint arXiv:1706.06083* (2017).
[29] W. Brendel, J. Rauber, M. Kümmerer, I. Ustyuzhaninov and M. Bethge, Accurate, reliable and fast robustness evaluation, *arXiv:1907.01003 [cs, stat]* (2019), arXiv: 1907.01003. http://arxiv.org/abs/1907.01003.
[30] J. Rony, L.G. Hafemann, L.S. Oliveira, I.B. Ayed, R. Sabourin and E. Granger, Decoupling Direction and Norm for Efficient Gradient-Based L2 Adversarial Attacks and Defenses, *arXiv:1811.09600 [cs]* (2019), arXiv: 1811.09600. http://arxiv.org/abs/1811.09600.
[31] M. Alzantot, Y. Sharma, S. Chakraborty, H. Zhang, C.-J. Hsieh and M. Srivastava, GenAttack: Practical Black-box Attacks with Gradient-Free Optimization, *arXiv:1805.11090 [cs]* (2019), arXiv: 1805.11090. http://arxiv.org/abs/1805.11090.
[32] F. Tramèr and D. Boneh, Adversarial Training and Robustness for Multiple Perturbations, *arXiv:1904.13000 [cs, stat]* (2019), arXiv: 1904.13000. http://arxiv.org/abs/1904.13000.
[33] J. Chen, M.I. Jordan and M.J. Wainwright, HopSkipJumpAttack: A Query-Efficient Decision-Based Attack, in: *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1277–1294–. ISSN 2375-1207. doi:10.1109/SP40000.2020.00045.
[34] M.-I. Nicolae, M. Sinn, M.N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. Molloy and B. Edwards, Adversarial Robustness Toolbox v1.2.0, *CoRR* **1807.01069** (2018). https://arxiv.org/pdf/1807.01069.
[35] J. Rauber, W. Brendel and M. Bethge, Foolbox: A Python toolbox to benchmark the robustness of machine learning models, in: *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017. http://arxiv.org/abs/1707.04131.
[36] J. Rauber, R. Zimmermann, M. Bethge and W. Brendel, Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX, *Journal of Open Source Software* **5**(53) (2020), 2607. doi:10.21105/joss.02607.

[37] X. Wang, J. Li, X. Kuang, Y.-a. Tan and J. Li, The security of machine learning in an adversarial setting: A survey, *Journal of Parallel and Distributed Computing* **130** (2019), 12–23. doi:10.1016/j.jpdc.2019.03.003.
[38] N. Papernot, P. McDaniel, X. Wu, S. Jha and A. Swami, Distillation as a defense to adversarial perturbations against deep neural networks, in: *2016 IEEE symposium on security and privacy (SP)*, IEEE, 2016, pp. 582–597.
[39] G.K. Dziugaite, Z. Ghahramani and D.M. Roy, A study of the effect of jpg compression on adversarial images, *arXiv preprint arXiv:1608.00853* (2016).
[40] N. Carlini and D. Wagner, Defensive distillation is not robust to adversarial examples, *arXiv preprint arXiv:1607.04311* (2016).
[41] J. Gao, B. Wang, Z. Lin, W. Xu and Y. Qi, DeepCloak: Masking Deep Neural Network Models for Robustness Against Adversarial Samples (2017), arXiv:1702.06763 [cs]. doi:10.48550/arXiv.1702.06763. http://arxiv.org/abs/1702.06763.
[42] G. Apruzzese, M. Colajanni and M. Marchetti, Evaluating the effectiveness of Adversarial Attacks against Botnet Detectors, in: *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, 2019, pp. 1–8. ISSN 2643-7929. doi:10.1109/NCA.2019.8935039.
[43] H. Qiu, T. Dong, T. Zhang, J. Lu, G. Memmi and M. Qiu, Adversarial Attacks Against Network Intrusion Detection in IoT Systems, *IEEE Internet of Things Journal* **8**(13) (2021), 10327–10335. doi:10.1109/JIOT.2020.3048038.
[44] M.J. Hashemi, G. Cusack and E. Keller, Towards Evaluation of NIDSs in Adversarial Setting, in *Big-DAMA '19*, Association for Computing Machinery, New York, NY, USA, 2019, pp. 14–21. ISBN 978-1-4503-6999-2. doi:10.1145/3359992.3366642.
[45] .
[46] A. Shafahi, W.R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras and T. Goldstein, Poison frogs! targeted clean-label poisoning attacks on neural networks, in: *NIPS*, 2018.
[47] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang and X. Zhang, Trojaning attack on neural networks, in: *NDSS*, 2002.
[48] Y. Vorobeychik and M. Kantarcioglu, Adversarial machine learning, *Synthesis Lectures on Artificial Intelligence and Machine Learning* **12**(3) (2018), 1–169.
[49] A. Serban, E. Poll and J. Visser, Adversarial Examples on Object Recognition: A Comprehensive Survey, *ACM Computing Surveys* **53**(3) (2020), 66:1–66:38. doi:10.1145/3398394.
[50] C. Xiao, J.-Y. Zhu, B. Li, W. He, M. Liu and D. Song, Spatially Transformed Adversarial Examples, *arXiv:1801.02612 [cs, stat]* (2018), arXiv: 1801.02612. http://arxiv.org/abs/1801.02612.
[51] J. Su, D.V. Vargas and K. Sakurai, One Pixel Attack for Fooling Deep Neural Networks, *IEEE Transactions on Evolutionary Computation* **23**(5) (2019), 828–841–. doi:10.1109/TEVC.2019.2890858.
[52] C. Guo, J.R. Gardner, Y. You, A.G. Wilson and K.Q. Weinberger, Simple Black-box Adversarial Attacks, *arXiv:1905.07121 [cs, stat]* (2019), arXiv: 1905.07121. http://arxiv.org/abs/1905.07121.
[53] E. Wong, F.R. Schmidt and J.Z. Kolter, Wasserstein Adversarial Examples via Projected Sinkhorn Iterations, *arXiv:1902.07906 [cs, stat]* (2020), arXiv: 1902.07906. http://arxiv.org/abs/1902.07906.
[54] S. Kotyan and D.V. Vargas, Adversarial Robustness Assessment: Why both L_0 and L_∞ Attacks Are Necessary, *arXiv:1906.06026 [cs, stat]* (2020), arXiv: 1906.06026. http://arxiv.org/abs/1906.06026.
[55] K. Grosse, D. Pfaff, M.T. Smith and M. Backes, The Limitations of Model Uncertainty in Adversarial Settings, *arXiv:1812.02606 [cs]* (2019), arXiv: 1812.02606. http://arxiv.org/abs/1812.02606.
[56] Y. Qin, N. Carlini, G. Cottrell, I. Goodfellow and C. Raffel, Imperceptible, Robust, and Targeted Adversarial Examples for Automatic Speech Recognition, in: *International Conference on Machine Learning*, PMLR, 2019, pp. 5231–5240–. ISSN 2640-3498. http://proceedings.mlr.press/v97/qin19a.html.
[57] X. Ji, Y. Cheng, Y. Zhang, K. Wang, C. Yan, W. Xu and K. Fu, Poltergeist: Acoustic Adversarial Machine Learning against Cameras and Computer Vision, IEEE Computer Society, 2021, pp. 1573–1588–. ISSN 2375-1207. ISBN 978-1-72818-934-5. doi:10.1109/SP40001.2021.00091. https://www.computer.org/csdl/proceedings-article/sp/2021/893400b573/1t0x9rMmOze.
[58] X. Liu, H. Yang, Z. Liu, L. Song, H. Li and Y. Chen, DPatch: An Adversarial Patch Attack on Object Detectors, *arXiv:1806.02299 [cs]* (2019), arXiv: 1806.02299. http://arxiv.org/abs/1806.02299.
[59] S.M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar and V.N. Venkatakrishnan, HOLMES: Real-time APT Detection through Correlationof Suspicious Information Flows, in: *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 1137–1152–.
[60] Q. Zou, X. Sun, P. Liu and A. Singhal, An Approach for Detection of Advanced Persistent Threat Attacks, *IEEE Computer Magazine* **53**(12) (2020), 92–96.
[61] C. Taylor, W. Harrison, A. Krings, N. Hanebutte and M. McQueen, Low-Level network attack recognition: a signature-based approach, *IEEE Proc. PDCS'2001* (2001), 570–574.
[62] S. Kaur and M. Singh, Automatic attack signature generation systems: A review, *IEEE Security & Privacy* **11**(6) (2013), 54–61.
[63] J. Choi, C. Choi, B. Ko and P. Kim, A method of DDoS attack detection using HTTP packet pattern and rule engine in cloud computing environment, *Soft Computing* **18**(9) (2014), 1697–1703.

[64] M. Amini, R. Jalili and H.R. Shahriari, RT-UNNID: A practical solution to real-time network-based intrusion detection using unsupervised neural networks, *computers & security* **25**(6) (2006), 459–468.
[65] O. Faker and E. Dogdu, Intrusion detection using big data and deep learning techniques, in: *ACMSE 2019 - Proceedings of the 2019 ACM Southeast Conference*, 2019, pp. 86–93. https://dl.acm.org/citation.cfm?id=3314439.
[66] K. Millar, A. Cheng, H.G. Chew and C.C. Lim, Deep learning for classifying malicious network traffic, in: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 11154 LNAI, 2018, pp. 156–161. http://link.springer.com/10.1007/978-3-030-04503-6_15.
[67] C. Yin, Y. Zhu, J. Fei and X. He, A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks, *IEEE Access* **5** (2017), 21954–21961. https://ieeexplore.ieee.org/abstract/document/8066291/.
[68] P. Mishra, V. Varadharajan, U. Tupakula and E.S. Pilli, A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection, *IEEE Communications Surveys Tutorials* **21**(1) (2019), 686–728–. doi:10.1109/COMST.2018.2847722.
[69] H. Zhang, Y. Li, Z. Lv, A.K. Sangaiah and T. Huang, A real-time and ubiquitous network attack detection based on deep belief network and support vector machine, *IEEE/CAA Journal of Automatica Sinica* **7**(3) (2020), 790–799–.
[70] H. Huang, J. Mu, N.Z. Gong, Q. Li, B. Liu and M. Xu, Data Poisoning Attacks to Deep Learning Based Recommender Systems, *Proceedings 2021 Network and Distributed System Security Symposium* (2021), arXiv: 2101.02644. doi:10.14722/ndss.2021.24525. http://arxiv.org/abs/2101.02644.
[71] X. Chen, C. Liu, B. Li, K. Lu and D. Song, Targeted backdoor attacks on deep learning systems using data poisoning, *arXiv preprint arXiv:1712.05526* (2017).
[72] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E.C. Lupu and F. Roli, Towards poisoning of deep learning algorithms with back-gradient optimization, in: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 27–38–.
[73] Abusing cloud services to fly under the radar, 2021, [Online; accessed 11. Nov. 2021]. https://research.nccgroup.com/2021/01/12/abusing-cloud-services-to-fly-under-the-radar.
[74] H.-Y. Lin and B. Biggio, Adversarial Machine Learning: Attacks From Laboratories to the Real World, *Computer* **54**(5) (2021), 56–60.
[75] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay and D. Mukhopadhyay, Adversarial Attacks and Defences: A Survey, *arXiv:1810.00069 [cs, stat]* (2018), arXiv: 1810.00069. http://arxiv.org/abs/1810.00069.
[76] Q. Cheng, S. Zhou, Y. Shen, D. Kong and C. Wu, Packet-Level Adversarial Network Traffic Crafting using Sequence Generative Adversarial Networks, *arXiv:2103.04794 [cs]* (2021), arXiv: 2103.04794. http://arxiv.org/abs/2103.04794.
[77] O. Ibitoye, O. Shafiq and A. Matrawy, Analyzing adversarial attacks against deep learning for intrusion detection in IoT networks, in: *2019 IEEE global communications conference (GLOBECOM)*, IEEE, 2019, pp. 1–6.
[78] J. Li, Y. Liu, T. Chen, Z. Xiao, Z. Li and J. Wang, Adversarial attacks and defenses on cyber-physical systems: A survey, *IEEE Internet of Things Journal* **7**(6) (2020), 5103–5115.
[79] JasonGerend, Set-DnsServerCache (DnsServer), 2022, [Online; accessed 21. Mar. 2022]. https://docs.microsoft.com/en-us/powershell/module/dnsserver/set-dnsservercache?view=windowsserver2022-ps.
[80] F. Pierazzi, F. Pendlebury, J. Cortellazzi and L. Cavallaro, Intriguing Properties of Adversarial ML Attacks in the Problem Space, in: *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1332–1349. ISSN 2375-1207. doi:10.1109/SP40000.2020.00073.
[81] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno and D. Song, Robust physical-world attacks on deep learning visual classification, in: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1625–1634.
[82] M. Sharif, S. Bhagavatula, L. Bauer and M.K. Reiter, A general framework for adversarial examples with objectives, *ACM Transactions on Privacy and Security (TOPS)* **22**(3) (2019), 1–30.