Deep Learning for Detecting Network Attacks: An End-to-end Approach

Qingtian Zou[1], Anoop Singhal[2], Xiaoyan Sun[3], and Peng Liu[1]

[1] The Pennsylvania State University
{qzz32,pxl20}@psu.edu
[2] National Institute of Standards and Technology
anoop.singhal@nist.gov
[3] California State University, Sacramento
xiaoyan.sun@csus.edu

Abstract. Network attack is still a major security concern for organizations worldwide. Recently, researchers have started to apply neural networks to detect network attacks by leveraging network traffic data. However, public network data sets have major drawbacks such as limited data sample variations and unbalanced data with respect to malicious and benign samples. In this paper, we present a new end-to-end approach to automatically generate high-quality network data using protocol fuzzing, and train the deep learning models using the fuzzed data to detect the network attacks that exploit the logic flaws within the network protocols. Our findings show that fuzzing generates data samples that cover real-world data and deep learning models trained with fuzzed data can successfully detect real network attacks.

Keywords: Network attack, Protocol fuzzing, Deep learning.

1 Introduction

Cyberattacks happen constantly with growing complexity and volume. As one of the most prevalent ways to compromise enterprise networks, network attack remains a prominent security concern. It can lead to serious consequences such as large-scale data breaches, system infection, and integrity degradation, particularly when network attacks are employed in attack strategies such as advanced persistent threats (APT) [13, 26]. Among the different types of network attacks, the *logic-flaw-exploiting network attacks*, which exploit the logic flaws within the protocol specifications or implementations, are very commonly seen. Detecting logic-flaw-exploiting network attacks is very important considering their common presence in APT campaigns. However, it is still a very challenging problem.

Network attack detection methods can mainly be classified into two categories: *host-independent* methods and *host-dependent* methods. The former solely relies on the network traffic, while the latter [4, 10, 21] depends on additional data collected on the victim hosts. The host-dependent methods have some evident drawbacks: they have fairly high deployment costs and operation costs; they are error-prone due to necessary manual configuration by human administrators. Therefore, host-independent detection methods are highly desired as they can decrease deployment and operation costs while reducing the attack surface of

detection system. Unfortunately, we found that the existing host-independent methods, including the classical intrusion detection approaches, often fall short in detecting some well-known and commonly used network attacks.

Recently there is a trend for using machine learning (ML) and deep learning (DL) techniques to detect network attacks. Nevertheless, the DL approaches could also achieve mixed results [8, 14], if they do not address the following two challenges. The first challenge is *useful data sets*. Neural networks require high-quality data and correct labels, which are hard to obtain in real world. Real-world network traffic is often flooded with benign packets, which makes labeling very difficult. Although public data sets [1, 7, 16, 19, 20] for network attacks are available, they are barely useful in detecting logic-flaw-exploiting network attacks due to unbalancing and different focuses. The second challenge is to *identify appropriate neural networks and train the models*. There are a variety of neural network architectures, including multi-layer perceptron (MLP), convolutional neural network (CNN), recurrent neural network (RNN), etc, which have different characteristics and capabilities. Questions such as which architecture works best for network attack detection, and how to tune the hyper-parameters within models for optimization, are not yet answered.

In this paper, we propose an end-to-end approach to detect the logic-flaw-exploiting network attacks. The end-to-end approach means it starts with acquiring data and ends with detecting attacks using the trained neural networks. To address the data generation challenge, we propose a new protocol fuzzing-based approach to generate the network traffic data. With protocol fuzzing, a large variety of *malicious* network packets for a chosen network attack can be generated at a fast speed. Since the network packets are all generated from the chosen network attacks, they can be labeled as malicious packets automatically without much human efforts. Protocol fuzzing can also generate data with more variations than real world data, or even data that are not yet observed in real world. Moreover, these merits remain when protocol fuzzing is leveraged to generate the needed *benign* network packets. It should be noted that our method is different from data synthesis. Data synthesis is to enhance existing data [11], while our method is to generate new data.

To address the neural network model training challenge, we propose the following procedures: 1) For network attacks (PtH) where we can identify fields of interest, we directly examine the data, and then propose the suitable data representation and neural network architecture. 2) For other network attacks that the field of interests are not obvious, such as DNS cache poisoning and ARP poisoning attacks, we apply different neural network architectures to find out the ones with best performance. We propose to use accuracy, F1 score, detection rate, and false positive rate as the metrics to evaluate the neural networks. All models are trained on the data set with fuzzing involved. We then select the models that work best and evaluate them further on both the fuzzing data set and real attack data set with no fuzzing involved.

The main contributions of this work include: 1) Proposing a DL based end-to-end approach to detect the logic-flaw-exploiting network attacks; 2) Proposing

protocol fuzzing to automatically generate high-quality network traffic data for applying DL techniques; 3) Proposing and evaluating neural network models for logic-flaw-exploiting network attack detection; 4) Demonstrating the effectiveness of our approach with three classical logic-flaw-exploiting network attacks, including PtH attack, DNS cache poisoning attack, and ARP poisoning attack.

2 Related Work

The research community has been tackling the network attack detection problem from different perspectives with both classical and novel approaches.

Traditional network attack detection approaches. Traditionally, people usually detect network attacks with approaches such as signature-based, rule-based, and anomaly detection-based methods. In the past, signature-based intrusion detection system (IDS) usually manually crafted signatures [22], which heavily depends on manual efforts. The current techniques focus more on automatic generation of signatures [12]. However, signatures need to be constantly updated to align with new attacks and signature-based detection can be easily evaded by slightly changing the attack payload. Similar problems also exist for rule-based methods [6], which constantly need updates to the rules. As for anomaly detection-based methods, although they require much less manual efforts for updating, they tend to raise too many false positives [3].

Traditional ML and DL for network attack detection. Network attacks are essential for APTs. Some common network attack types include probing, DoS, Remote-to-local, etc. Both traditional ML and DL methods have been adopted for network attack detection. Some focus on one type of network attack and perform binary classifications. For example, MADE [18] employs ML to detect malware C&C network traffic, Ongun et al. [17] employs ML to detect botnet traffic, and DeepDefense [24] employs DL to detect distributed DoS (DDoS) attacks. Others [8, 14, 15, 23, 25] try multi-class classifications, which include one benign class and multiple malicious classes for different kinds of network attacks. The above-mentioned research works all use public data sets.

Network data sets for training and testing detection models. To apply DL for network attack detection, a data set is required. Commonly used public data sets include KDD99 [19], NSL-KDD [7], UNSW-NB15 [16], CICIDS2017 [20], and CSE-CIC-IDS2018 [1]. The public data sets are all generated in test-bed environments, with simulated benign and malicious activities. These data sets are often unbalanced due to overwhelming amount of benign data. Even for only malicious activities, multiple types of attacks may be included and the amount of malicious data for each attack type varies a lot. However, balanced data set is important for DL and it is very difficult to label the unbalanced data. Moreover, these data sets focus on network attacks that do not exploit logic flaws, such as DDoS, worms, and C&C over HTTP/HTTPS. They do not contain data about PtH attack, DNS cache poisoning attack, and ARP poisoning attack, which are our detection targets. Though related protocols (e.g. DNS and ARP) are included in those public data sets, such network packets are generated as side effects of other activities, but not because those data sets intentionally

want to include such data or launching attacks using those protocols. In a word, existing public data sets are useless in our work.

Protocol fuzzing. Fuzzing is originally a black-box software testing technique, which reveals implementation bugs by feeding mutated data. A key function of fuzzers is to generate randomized data which still follows the original semantics. There are tools for building flexible and security-oriented network protocol fuzzers, such as SNOOZE [5]. Network protocol fuzzing frameworks such as AutoFuzz [2, 9] were also presented. They either act as clients, constructing packets from the beginning, or act as proxies, modifying packets on the fly. We use protocol fuzzing for a different purpose to directly generate high-quality data sets for training neural networks. Instead of using the tools/frameworks mentioned earlier, we prepare our own fuzzing scripts for this specific purpose.

3 Experiment Setup

Since the available public data sets are barely useful for detecting the logic-flaw-exploiting network attacks, this paper will generate comprehensive data sets from scratch, including benign and malicious data sets. We have performed data generation for all three demonstration attacks including PtH, DNS cache poisoning, and ARP poisoning. ARP poisoning attack only requires one malicious packet for a successful attack, so we call it the single-packet attack. PtH and DNS cache poisoning attacks, however, need multiple malicious packets for one successful attack, so we call them multi-packet attacks. Due to page limits, we only discuss data generation details about multiple-packet attacks in this section because they are more complicated than single-packet attacks. Below subsections discuss the general approach and implementation principles of protocol fuzzing followed by attack-specific details. All attacks are carried out thousands of times so that a fair amount of malicious data can be collected. Benign data generation also lasts long enough to gather the commensurate amount of data compared to malicious data. The network packet capturing is performed at the victim's side. After that, information about detection neural networks are provided.

3.1 Protocol Fuzzing and The Implementation

In client-server enterprise computing, the server-side protocol implementations are often complex and error-prone. Hence, there is a need to achieve thorough testing of the server-side implementation. Protocol fuzzing tools [2, 5, 9] are usually functioning at the client side, so that unexpected errors on the tested server programs may be triggered. A main difference between protocol fuzzing and software fuzzing is that the protocol specification, especially its state transition diagram, will be used to guide the fuzzing process. In this way, fuzzing tests could be performed in a stateful manner.

This paper leverages protocol fuzzing to change the contents of network packets, specifically, the values of some fields in the packets. If a field is to be fuzzed, it will be assigned with pre-determined values, rather than the values chosen by the network client program. The fuzz fields are chosen based on the following steps: 1) All fields in the packet of the attack-specific protocol are considered. 2) One field on the list will be picked and fuzzed by assigning pre-determined

values, rather than values that are normally provided by the network programs. 3) The success rate of the attack after fuzzing the field will be monitored. If the attack success rate is above 50%, it confirms that this field can be fuzzed. 4) After one field is fuzzed, the above steps will be repeated for the next field on the list, while keeping the already fuzzed field(s) still fuzzed.

Result: `BList`, which stores fields to fuzz
input `AList` of all available fields;
initialize an empty `BList` to store fields to fuzz;
foreach *`field` in `AList`* **do**
  fuzz `field`;
  fuzz all fields in `BList`;
  launch the attack for hundreds of times;
  count successful attacks and calculate success rate;
  if *attack success rate is over 50%* **then**
    add `field` to `BList`;
  end
end

Algorithm 1: Select fields to be fuzzed.

To ensure the fuzzed packets are valid, we need to firstly make sure *AList*, the list of all candidate fuzzing fields, does not contain fields that will affect the packets' integrity, such as fields of checksum values and packet lengths. The values of those fields should not be arbitrarily changed. Furthermore, when we choose the fields to be fuzzed, we need to make sure the attack success rate after fuzzing this field is always above 50%.

An additional benefit of protocol fuzzing is that it can generate and cover malicious data samples which may otherwise be overlooked when applying deep learning. In deep learning, the changed values for the fuzzing fields may make the malicious data samples misclassified as benign. With protocol fuzzing, if the malicious data are generated in attacks, they'll be labeled as malicious automatically. Thus, these malicious data samples won't be omitted in the malicious data set.

3.2 PtH

PtH Attack. PtH is a well-known technique for lateral movement. In remote login, plain text passwords are usually converted to hashes for authentication. Some authentication mechanisms only check whether hashes or the calculation results of them matches or not. PtH relies on these vulnerable mechanisms to impersonate normal users with dumped hashes. We assume that: (a) normal users use benign client programs that are usually authenticated through more reliable mechanisms other than just using hashes, and that (b) attackers cannot get the plain text passwords and have to rely on hashes to impersonate a normal user. We can capture the network packets at the server side and find out which kind of authentication mechanism is used by a user: the more reliable mechanism, or the vulnerable mechanism using only hashes. The login sessions using those vulnerable authentication mechanisms can then be identified as PtH attack.

Windows remote login processes, if not properly configured, can use such vulnerable authentication mechanisms. Windows remote login can be divided

Table 1: Fields of interest.

Layer	Fields	Size in bytes	Explanation
ETH	Dst_MAC	6	Destination MAC address
	Src_MAC	6	Source MAC address
	ETH_type	2	Indicate which protocol is encapsulated in the payload of the frame.
ARP	HTYPE	2	Network link protocol type. For Ethernet, this field is 1.
	PTYPE	2	For IPv4, this value should always be 0x0800.
	HLEN	1	Length of a hardware address. For Ethernet addresses, the length is 6.
	PLEN	1	For IPv4 addresses, this value should always be 4.
	OpCode	2	Specifies the operation that the sender is performing: 1 for request, 2 for reply.
	SHA	6	Source hardware (MAC) address.
	SPA	4	Source internetwork (IP) address.
	THA	6	Target hardware (MAC) address.
	TPA	4	Target internetwork (IP) address.
IP	Version	4/8	For IPv4, this is always equal to 4.
	IHL	4/8	Internet header length.
	DSF	1	Differentiated service field, which includes differentiated services code point and explicit congestion notification.
	TLen	2	The entire packet size in bytes.
	ID	2	Identification field.
	Flags	3/8	3 bits for controling or identifying fragments.
	FragOff	13/8	Fragment offset.
	TTL	1	Time to live field, which limits a datagram's lifetime.
	prot	1	This field defines the protocol used in the data portion of the IP datagram.
	chksum	2	Header checksum for error-checking of the header.
	src_add	4	Source IPv4 address.
	dst_add	4	Destination IPv4 address.
UDP	src_port	2	Source port number.
	dst_port	2	Destination port number.
	hd_len	2	The length in bytes of the UDP header and UDP data.
	chksum	2	Checksum field for error-checking of the UDP header and UDP data.
DNS	TID	2	Transaction ID.
	flags	2	Control flags
	q	2	The number of entries in the question section.
	AnRR	2	The number of resource records in the answer section.
	AuRR	2	The number of resource records in the authoritative section.
	AdRR	2	The number of resource records in the additional section.
SMB/ SMB2	cmd	2	The command code of this packet.
	flags	4	Indicate how to process the operation
	NT_status	4	Status or error code.

into three stages, protocol and mechanism negotiation (initial communication), authentication, and task-specific communication (afterwards communication). Each stage contains multiple network packets, and hashes are used in the authentication stage for impersonation. The authentication stage can be viewed as a sequence made up of client's authentication request, server's challenge, client's challenge response and server's authentication response. The client first sends a session setup request to the server; then the server responds to the client with a challenge; on receiving the challenge, the client uses the challenge and hashes to do calculations and sends back the result in challenge response packet; finally, the server verifies the result and sends back authentication response indicating whether authentication succeeds or not.

Data generation. We set up a Windows 2012 Server R2 as the victim server machine, a Windows 7 as the user client machine, and another Kali Linux as the attacker machine. The data sets are automatically generated by protocol fuzzing, and the protocol of interest here is Server Message Block (SMB), or a newer version of it, denoted as SMB2. SMB/SMB2 provides functions including file sharing, network browsing, printing, and inter-process communication over a network. In our data generation, more than 15 fields are fuzzed in each SMB/SMB2 packets, including `SMB flags`, `SMB capabilities`, and fields in SMB header, etc. e leverage the PtH script in Metasploit Framework to launch the attack. Boxes connected with solid lines are what happens at foreground, and boxes in the dash line area happen behind the scene. The process is to start the Metasploit Framework, set exploit parameters, start the exploitation, and then wait 25 seconds while monitoring the attack status. If the waiting time is too short, the attack may be stopped before completion. While the console is

waiting at the foreground, the exploitation is ongoing at the background. Network packets in all the three stages, initial communication, authentication, and afterwards communication, are fuzzed. After the exploitation, based on whether the attack succeeds or not, we may continue to establish C&C, like what a real attacker will do. (The C&C network traffic are mainly TCP packets, which are not used for attack detection. Details are discussed later.) Finally, we quit all possibly established sessions and the Metasploit Framework, and then either freshly start another fuzzing iteration to generate more data or stop. The sign of a successful PtH attack is an established reverse shell, which can be observed at the attacker's side.

The same fuzzing method has also been applied in the generation of benign data. We first prepare a list of normal commands, including files reading, writing, network interactions, etc. For each benign fuzzing iteration, we randomly choose a command from the list, and then use valid username, plain-text password, and tool to log in to the server and execute the command.

All the network packets from malicious and benign network traffic are captured using Wireshark at the victim's side. Due to fuzzing, not all PtH attempts or benign access attempts can be guaranteed to succeed. For failed PtH attempts, we remove them from malicious data because they do not generate real malicious impact, and they cannot be categorized as benign either because they are generated with attacker tools for malicious purpose. For failed benign accesses, we keep them in benign data, because normal users can also have failed logins due to typos, wrong passwords, etc.

In one PtH attack, there are packets for initial communications, authentication and afterwards communications. One data sample consists of multiple packets, and those packets may come from one, two, or all of the three stages above. Besides, one complete PtH attack or benign activity most certainly contains more packets than one data sample can represent. When labeling, if the session is malicious, then all data samples generated from this session is labeled malicious, and the same is also true for the benign cases.

Detections. To detect PtH attack with neural networks, we have two key insights that help determine the representation of data samples: 1) Network communication for authentication is actually a sequence of network packets in certain order. An earlier packet can affect the packet afterwards. For example, the first several packets between a server and a client may be used to communicate and determine which protocol to use (e.g. SMB or SMB2), and packets afterwards will use the decided protocol. The attack is to get authenticated by the server, which requires a sequence of packets to accomplish. Therefore, each data sample should be a sequence of packets, rather than an individual packet. 2) PtH relies on authentication mechanisms that legitimate users usually don't use. The network packets for the benign and malicious authentication are different. Since both authentication methods use SMB/SMB2 packets, the differences between them thus exist in the fields of the SMB/SMB2 layer. Therefore, data in `SMB/SMB2` layer is used for PtH detection. In addition, the differences of field values between benign and malicious authentication will be helpful to distin-

guish them. For this attack, we choose Long-short term memory (LSTM) as the architecture for the neural network.

3.3 DNS Cache Poisoning

DNS cache poisoning. A major functionality of DNS is to provide the mapping between the domain names and IP addresses. When a client program refers to a domain name, the domain name needs to be translated to an IP address. The DNS servers are responsible to perform such translation.

The global DNS system has a hierarchical structure that contains root name servers, top-level domain name servers, and authoritative name servers. Some examples are the public DNS servers `8.8.8.8` and `8.8.4.4` provided by Google, and recently released `1.1.1.1` by Cloudflare. These name servers, referred as the global DNS servers, provide records that maps the domain names and IP addresses. Due to the geological distance between user machines and the global DNS servers, it is very costly to contact the global DNS servers every time very often. To reduce the cost, organizations deploy their own DNS servers, referred as local DNS servers, within the LAN to cache the most commonly used mappings between domain names and IP addresses. Generally, when a user machine needs to make connection with a destination machine, it will contact the local DNS server first to resolve the domain name. If the local DNS server does not cache the DNS record for this domain name, it will send out a DNS query to the global DNS server to get the answer for the user machine. The user machine gets to know the IP address after receiving the response.

DNS cache poisoning attack can target local DNS servers. When the local DNS server receives a query which it does not have the corresponding records (first stage), it will inquire the global DNS server (second stage). On receiving the response (third stage), the local DNS server saves this record in its cache to avoid inquiring the global DNS again when receiving the same query. It then forwards the response to the user machine (fourth stage). However, the DNS server cannot verify the response at the third stage, and this is where the attacker can fool the local DNS server. Pretending as the global DNS server, the attacker can send a spoofed DNS response to the local DNS server with falsified DNS records. If the fake response arrives earlier than the real one, the local DNS server will save the falsified record to its cache and forward it to the user machine. When new queries about the same domain name comes in, the local DNS server will not query the global DNS server again because the corresponding record has been cached. Consequently, it will answer the user machine with the falsified record, until the record expires or the cache is flushed.

Data generation. For this attack, ten fields, such as `time to live` values in different layers, are fuzzed. The test bed contains three machines: a local DNS server whose DNS cache is flushed periodically, a user machine which sends out DNS queries to the local DNS server periodically, and an attacker machine which sniffs for DNS requests sent by the local DNS server and answers them with spoofed responses as in the attack scenario, or does nothing otherwise.

In the malicious scenario, we make the user machine ask for the IP address of one specific domain name from the local DNS server using command *dig*. The

domain name is one that does not have a corresponding record on the local DNS server, thus enabling the DNS cache poisoning attack towards it. The attacker machine sniffs for DNS queries with that specific domain name sent out from the local DNS server, and responds them with fuzzed DNS responses with falsified IP addresses. Then the DNS cache gets poisoned and the user machine gets the falsified DNS record. We keep the user machine sending out DNS queries periodically, so that the above process repeats many times and a large amount of data can be generated. However, as discussed earlier, if the local DNS server has the record for the domain name in its cache, it will not send out DNS queries for it. This is why we flush the DNS cache of the local DNS server, so that it remains vulnerable in different iterations. If the attack is successful, the falsified IP addresses can be seen on the results of *dig*.

In the benign scenario, we prepare a list containing 4098 domain names. In each iteration, the user machine randomly chooses one domain name from the list, and sends a request to the local DNS server. To resemble the malicious scenario, the cache of local DNS server is also flushed periodically so that the local DNS server always needs to communicate with the global DNS server.

The domain name used in the malicious scenario and the domain names used in the benign scenario do not overlap. Both the domain names and the IP addresses (falsified or genuine) are excluded during training, which can be treated as signatures. Because DNS cache poisoning is a multi-packet attack, the labeling to data samples is also based on sessions, similar to PtH attack.

Detections. Network packets from DNS cache poisoning attack form sessions which consist of queries and answers. Therefore, each data sample should include data from multiple network packets. In addition, it is not clear which fields may be of importance, so we need to investigate the packet content, rather than simply generalizing the packets with packet types as we did in PtH detection. The data samples are processed to be image-like. That is, each row represent one packet, and each element in the row represent one byte in that packet. We use a convolutional neural network (CNN) to do the classifications, which has been proven to work well in image classification problems. The labeling is done towards each data sample, which is the entire matrix, rather than an individual packet. During the data processing process, the malicious and benign data are processed separately. Matrices generated from malicious data are labeled as malicious, and matrices from benign data are labeled as benign. Similar to PtH detection, we have trained a series of neural networks with different neural network hyper-parameters and data samples of different window sizes and window steps. That means we can adjust the number of packets k included in each data sample and thus change the size of matrix.

4 Evaluations

This section provides the evaluation results of the three demonstration attacks on the selected best-performing and best-detecting models. For comparison with DL models, we have also trained traditional ML models, including k-nearest neighbor (kNN) models, support vector machine (SVM) models with various kernels, decision tree (DT) models, and random forest (RF) models. They are

trained, selected, and evaluated on the same data sets. For PtH and ARP poisoning, the traditional ML models' data samples and features are the same as those for DL models. However, for DNS cache poisoning, the same data sample and feature cannot be used because the input space is too large for traditional ML models to handle. Therefore, we employed principal component analysis (PCA) for dimension reduction, and only select the top-rated one-fifth PCA features. On average, they can explain about 97.09% of the original data.

4.1 Model Selection

For model selection, we consider not only the perspective of neural network performance, but also the perspective of security. We use accuracy (*Acc* and F1 score (*F1*), two commonly used metrics, to measure the classification, and use detection rate (*DR*) and false positive rate (*FPR*) for attack detection effectiveness. Assuming the numbers of true positives, true negatives, false positives and false negatives are presented as TP, TN, FP, FN, respectively, then $Acc = (TP+TN)/(TP+TN+FP+FN), F1 = TP/(TP+0.5*(FP+FN)), DR = TP/(TP+FN)$, and $FPR = FP/(TN+FP)$. *DR* shows the detector's ability of detecting attacks. *FPR* shows how likely the detector raises false alarms. We call the best-performing model as the one that gets the highest average of *Acc* and *F1*, denoted as $P = \frac{Acc+F1}{2}$, and the best-detecting model as the one that gets the highest average of *DR* and $1-FPR$, denoted as $D = \frac{DR+1-FPR}{2}$. If FPR cannot be calculated (no benign data sample), we let $D = DR$. We simply take the average because all the chosen metrics are equally important for evaluations.

The generated fuzzing data set is randomly split into two parts: 80% as the training set, and 20% as the test set. The training set is then further randomly split into four parts of about the same size, upon which 4-fold cross-validation is employed to avoid over-fitting. All the reported results are the average results among four folds. The best-performing and best-detecting models are selected based on the average P and D results on the validation set across all four folds.

4.2 Data Sets

Table 2 shows the data set statistics. The data set contains fuzzed set (split into training set and test set) and non-fuzzed set (real attack set). A data set with sufficient and balanced data samples is essential for training the models effectively. Lack of training data can result in poor results, while biased data sets may result in biased models. If the fuzzing data set is already balanced, we directly use all the data samples without balancing. Otherwise, we perform data set balancing first. Specifically, if the benign data sets have significantly more data samples than the malicious data sets, we down-sample the benign data sets to match the size of malicious data sets, and vice versa.

Table 2: Data set statistics.

Attacks	Set	Size	Benign to malicious ratio
ARP poisoning	Training	9584	1.005:1
	Test	2400	0.982:1
	Real attack	17471	0:1
PtH* (best-performing)	Training	3932	1.364:1
	Test	983	1.329:1
	Real attack	214	0:1
PtH* (best-detecting)	Training	2556	0.974:1
	Test	640	0.839:1
	Real attack	192	0:1
DNS cache poisoning*	Training	30928	1.003:1
	Test	7732	0.988:1
	Real attack	263	0:1

* For multi-packet attacks, we only list the data set statistics corresponding to the best-performing or best-detecting models.

4.3 Best-performing Models

Table 3 presents the evaluation results on the best-performing models for each network attack. All models get acceptable to good results on training set and test set. **For multi-packet attacks, DL models are substantially better than traditional ML models, especially on real attack set.** In PtH detection, the LSTM model achieves near 99% accuracy on the real attack set, while ML models cannot reach 1/4 accuracy. In DNS cache poisoning detection, the CNN model's accuracy on the real attack set is 100%, while ML model can reach about 47% accuracy at most. Selected DL models' F1 scores are also far better than those of traditional ML models. For ARP poisoning detection, DL models do not have many advantages over traditional ML models, and all models' performances downgrade on real attack set comparing to those of training set and test set. The reason is that the real attack set for ARP poisoning is generated on a different LAN, with different valid MAC and IP addresses.

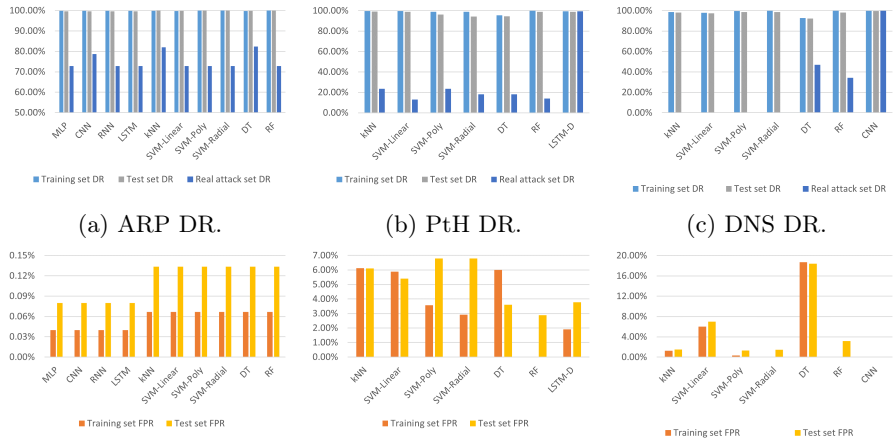
Table 3: Evaluation results on best-performing models.

Attacks	DL or ML	Model type[1]	Training set		Test set		Real attack set	
ARP	DL	MLP	99.91%	0.9991	99.75%	0.9975	72.84%	0.8429
		CNN	99.94%	0.9994	99.79%	0.9979	73.02%	0.8441
		RNN	99.91%	0.9991	99.75%	0.9975	72.83%	0.8428
		LSTM	99.91%	0.9991	99.75%	0.9975	72.83%	0.8428
	ML	kNN	99.90%	0.9990	99.93%	0.9993	81.99%	0.9010
		SVM-Linear	99.87%	0.9987	99.90%	0.9990	72.83%	0.8428
		SVM-Poly	99.96%	0.9996	99.93%	0.9993	72.83%	0.8428
		SVM-Radial	99.97%	0.9997	99.93%	0.9993	72.83%	0.8428
		DT	99.84%	0.9984	99.90%	0.9990	82.35%	0.9032
		RF	99.97%	0.9997	99.93%	0.9993	72.83%	0.8428
PtH	DL	LSTM-P	98.45%	0.9865	98.07%	0.9831	98.96%	0.9948
	ML	kNN	96.77%	0.9682	96.53%	0.9658	23.44%	0.3797
		SVM-Linear	96.89%	0.9694	96.72%	0.9674	13.02%	0.2304
		SVM-Poly	97.75%	0.9779	94.69%	0.9479	23.44%	0.3797
		SVM-Radial	98.07%	0.9810	93.72%	0.9378	18.23%	0.3084
		DT	94.70%	0.9467	95.44%	0.9533	18.23%	0.3084
		RF	100.00%	1.0000	97.99%	0.9798	14.06%	0.2466
DNS	DL	CNN	99.87%	0.9987	99.73%	0.9973	100.00%	1.0000
	ML	kNN	98.67%	0.9867	98.35%	0.9834	0.00%	0.0000
		SVM-Linear	96.01%	0.9608	95.17%	0.9527	0.00%	0.0000
		SVM-Poly	99.63%	0.9963	98.70%	0.9870	0.00%	0.0000
		SVM-Radial	100.00%	1.0000	98.66%	0.9867	0.00%	0.0000
		DT	87.01%	0.8771	86.88%	0.8754	47.01%	0.6395
		RF	100.00%	1.0000	97.50%	0.9752	34.19%	0.5096

[1] For multi-packet attacks, only proposed DL models are presented.

4.4 Best-detecting Models

Figure 1 presents the evaluation results of best-detecting models. FPRs on real attack sets are not presented because there is no negative data sample, so FPR



(a) ARP DR. (b) PtH DR. (c) DNS DR.

(d) ARP FPR. (e) PtH FPR. (f) DNS FPR.

Fig. 1: Evaluation results on the best-detecting models.

cannot be calculated. Similar to the best-performing case, all models get acceptable to good results on training and test set. For single-packet attack detection, DL models do not have many advantages over ML models. **For multi-packet attacks, DL models are better than ML models, especially on real attack set.** Because there is no negative data sample in the real attack set, $DR = Acc$. As for FPR, although it cannot be calculated in the real attack set, results show that DL models achieve generally lower FPRs comparing to ML models on the training and test sets.

5 Discussions and Limitations

Lack of efficiency: Training a neural network requires a large amount of data samples. However, the number of data samples can be affected in many ways. For example, protocol fuzzing in nature cannot guarantee that all malicious/benign activities are successful. Although the fuzzed values are in a valid range, the network packets with fuzzed values may still get rejected by the server or trigger some unexpected circumstances, leading to an interrupted session. Those data are probably useless as discussed in section 3.2. Also, the removal of duplicate (same data in one class) and double-dipping (same data among different classes) data samples will also affect the number of data samples. In a word, not all collected data can be used as data sample for neural network training.

Another factor that affects the efficiency is the time consumed by each benign/malicious activity. Except for some simple activities like MAC-IP address resolving with only several ARP packets, other complicated activities need time to carry out, especially those containing hundreds or more network packets. Moreover, depending on the mechanism of packet processing, the client/server may also need more time before it can respond. For example, in PtH data generation, one successful attack contains 300 to 400 packets (and not all of them are usable to generate data sample), and some time intervals between adjacent packets can be as large as 0.5 second. In addition, in our experiments, we manually inserted idle time intervals. This time interval is reserved so that the

exploitation can continue to run to reach a successful end. If this time interval is removed or too short, then the attack process is very likely to end in the middle of exploitation. In a word, each data generation iteration takes time to complete.

As a result, our data generating efficiency is not very high. Take PtH as an example, we spent about 4 days running 5,000 attack iterations, of which 611 failed. The total amount of network packets captured is 497,956, of which 103,718 are related packets. However, the final number of data samples is only in the thousands, as shown in Table 2.

Neural networks for various network attacks: Though we have verified our idea on three chosen network attacks, we trained separate neural networks for different attacks. We can not train a generic neural network to detect various network attacks. It is difficult to train such a neural network because different network attacks have different characteristics, which may need different data representations and neural network architectures.

Impact of probability threshold: The raw outputs for output layers of the detection neural networks are the probabilities for the data sample to be benign or malicious, which add up to 1. The raw outputs can be converted to classification results. If the probability for malicious class is beyond a threshold (e.g., 0.5), then the data sample is classified as malicious. Otherwise, it is classified as benign. When the probability threshold increases, the model is more likely to classify a data sample as benign, and thus decrease detection rates and false positive rates. The probability threshold can be tuned depending on whether the defender prefers higher detection rates or lower false positive rates.

6 Conclusion

This paper presents an end-to-end approach to detect the logic-flaw-exploiting network attacks using DL. The end-to-end approach begins with data generation and collection, and ends with attack detection with neural networks. We address two major challenges in applying DL for logic-flaw-exploiting network attack detection: the generation of useful data sets and the training of appropriate neural network models. We show the effectiveness of our approach with three specific demonstration attacks, including PtH, DNS cache poisoning, and ARP poisoning. We have generated high quality network traffic data using protocol fuzzing, trained neural networks with generated data, and evaluated the trained models from the perspective of both neural network performance and attack detection. We have also discussed the limitations of our experiments and approach.

Disclaimer

References

1. IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB (Jan 2020), `https://www.unb.ca/cic/datasets/ids-2018.html`, [Accessed Jul 4 2020]

2. Aitel, D.: The advantages of block-based protocol analysis for security testing. Immunity Inc., February **105**, 106 (2002), http://www.immunityinc.com/downloads/advantages_of_block_based_analysis.pdf
3. Amini, M.e.a.: Rt-unnid: A practical solution to real-time network-based intrusion detection using unsupervised neural networks. computers & security **25**(6), 459–468 (2006)
4. Arote, P., Arya, K.V.: Detection and prevention against arp poisoning attack using modified icmp and voting. In: 2015 International Conference on Computational Intelligence and Networks. IEEE (2015)
5. Banks, G.e.a.: SNOOZE: Toward a Stateful NetwOrk prOtocol fuzZEr. In: Springer, pp. 343–358 (2006)
6. Choi, J.e.a.: A method of ddos attack detection using http packet pattern and rule engine in cloud computing environment. Soft Computing **18**(9), 1697–1703 (2014)
7. Dhanabal, L., Shantharajah, S.: A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. International Journal of Advanced Research in Computer and Communication Engineering **4**(6), 446–452 (2015)
8. Faker, O., Dogdu, E.: Intrusion detection using big data and deep learning techniques. In: ACMSE 2019 - Proceedings of the 2019 ACM Southeast Conference (2019)
9. Gorbunov, S., Rosenbloom, A.: AutoFuzz: Automated Network Protocol Fuzzing Framework. International Journal of Computer Science and Network Security **10**(8), 239–245 (2010)
10. Goswami, S.e.a.: An unsupervised method for detection of xss attack. IJ Network Security **19**(5), 761–775 (2017)
11. Jan, S.T.e.a.: Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In: The 41st IEEE Symposium on Security and Privacy (IEEE SP) (2020)
12. Kaur, S., Singh, M.: Automatic attack signature generation systems: A review. IEEE Security & Privacy **11**(6), 54–61 (2013)
13. Milajerdi, S.M.e.a.: Holmes: Real-time apt detection through correlationof suspicious information flows. In: 2019 IEEE Symposium on Security and Privacy (SP). IEEE (2019)
14. Millar, K.e.a.: 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)
15. Mishra, P.e.a.: A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Communications Surveys Tutorials **21**(1), 686–728 (2019). https://doi.org/10.1109/COMST.2018.2847722
16. Moustafa, N., Slay, J.: UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings. Institute of Electrical and Electronics Engineers Inc. (dec 2015)
17. Ongun, T.e.a.: On designing machine learning models for malicious network traffic classification. arXiv:1907.04846 [cs, stat] (Jul 2019), http://arxiv.org/abs/1907.04846, arXiv: 1907.04846
18. Oprea, A.e.a.: Made: Security analytics for enterprise threat detection. In: Proceedings of the 34th Annual Computer Security Applications Conference. ACSAC '18, Association for Computing Machinery (Dec 2018). https://doi.org/10.1145/3274694.3274710
19. Pfahringer, B.: Winning the kdd99 classification cup: bagged boosting. ACM SIGKDD Explorations Newsletter **1**(2), 65–66 (2000)

20. Sharafaldin, I.e.a.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy. vol. 2018-Janua (2018)
21. Sun, H.M.e.a.: Dependns: Dependable mechanism against dns cache poisoning. In: International Conference on Cryptology and Network Security. pp. 174–188. Springer (2009)
22. Taylor, C.e.a.: Low-level network attack recognition: a signature-based approach. IEEE Proc. PDCS'2001 (2001)
23. Yin, C.e.a.: A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. IEEE Access **5**, 21954–21961 (2017)
24. Yuan, X., Li, C., Li, X.: DeepDefense: Identifying DDoS Attack via Deep Learning. In: 2017 IEEE International Conference on Smart Computing, SMARTCOMP 2017 (2017)
25. Zhang, Y.e.a.: PCCN: Parallel Cross Convolutional Neural Network for Abnormal Network Traffic Flows Detection in Multi-class imbalanced Network Traffic Flows. IEEE Access pp. 1–1 (2019)
26. Zou, Q.e.a.: An approach for detection of advanced persistent threat attacks. IEEE Annals of the History of Computing **53**(12), 92–96 (2020)