Probing the Machine Learning Network Defense Models

Tam Nguyen
North Carolina State University
Raleigh, North Carolina
tam.nguyen@ncsu.edu

Sri Sai Bhargav Tiruveedhula
North Carolina State University
Raleigh, North Carolina
stiruve@ncsu.edu

Omkar Parkhe
North Carolina State University
Raleigh, North Carolina
ojparkhe@ncsu.edu

ABSTRACT

Machine learning is gaining traction in the network security domain as the solution to multiple problems such as the scalability of network administration, security play-book automation, cost reduction for large scale companies, etc. Compromising machine learning model is then a very desirable goal for adversaries. Previous works have been done on adversarial machine learning but not on models being deployed as a network security solution. For the first time, this paper will present vivid illustrations into the steps of attacking a machine learning based network security model within a reputable open-source framework named Stratosphere. The main contribution is our white-box testing algorithms that we used in order to successfully "break" Stratosphere. The algorithms can also be used by other researchers to evaluate the base-line robustness of their machine learning designs.

CCS CONCEPTS

• **Security and privacy** → *Firewalls*;

KEYWORDS

machine learning, adversarial, network, firewall, cybersecurity

1 INTRODUCTION

There is a significant gap between the amounts of connected devices and the number of cyber security professionals. Per U.S. Bureau of Labor Statistics [7], a projected growth in cyber security jobs from 2014 to 2024 is 18% while Cisco [10] predicted a 100% increase in network-enabled devices, growing from 4 billions in 2016 to 8 billions units in 2021. Consequently, global data traffics will increase by at least 5 times. In order to overcome cybersecurity shortages, to automate security play-book, and to reduce operational costs at scale, more and more companies deploy Machine Learning (ML) based security solutions. For example, a global corporation like Google is using Software Defined Network to effectively "link" their data centers located around the world, ML models can then be deployed to effectively monitor, correlate and control network traffics 24/7.

While being very productive at scale, ML models come with their own sets of problems. For example, Artificial Neural Networks (ANNs) are fit for detecting non-linear anomalies but they tend to suffer from local minima leading to long learning time. As the number of features increases, ANNs take even more time to learn. Such problems are usually not communicated well in research works or in commercial products due to various reasons.

The main contribution of this paper is our systematic process of attacking a ML model. Our white-box methodology includes two functions: Explore and Blend through which we abuse the detection scores given by the models. We were able to reach the rate of 100% model evasion (zero detection). Through this work, we make a case for the need of a more secure development process in ML based network security solutions.

The structure of our paper starts with a discussion on the inherent problems within common ML algorithms, to be followed by several theoretical methods to clone ML models (section 2). Per ML cyber kill chain [18], the cloning of ML models, while does not do any damage, is still considered as a form of attack (reconnaissance attack). Our white-box attack methods presented in this paper are completely different from cloning attacks. However, those two types of attacks are strongly related - hackers would need to clone a model first in order to perform the kind of white-box attacks to be presented in this paper.

We then provide background information on StratosphereIPS (section 3) and our methodology on attacking their models (section 5). Given a known malicious flow, "Explore" identifies which part of the flow contributes most to the detection score. In an analogy, if the text of the US National Anthem can be identified by "Oh say, can you see", then "Oh say" is the part that contributes most to the detection score. Once identified, the "Blend" function will dilute just the identified part, leading to weaken detection score or even a complete model evasion.

We will also show the detailed results that we gathered (section 6 and our recommendations based on what we learned (section 7).

2 RELATED WORKS

We found it crucial to understand that there are inherent weaknesses in each machine learning model and even when everything is perfect, there is always a chance for a particular model to be cloned or reverse-engineered.

2.1 Inherent problems with ML models

Based on an earlier survey done by Buczak and Guven [6], most common ML models come with their own pros and cons. By design, **Artificial Neural Networks (ANN)** are fit for non-linear problems but tend to suffer from local minima leading to long learning time, and as the number of features increases, the longer it takes to learn. The performance of ANNs can also be very inconsistent. While they can identify 100% of the normal behavior, the amount of false alarms may sometimes reach 76% depending on what kind of attacks were being executed.

Bayesian network is a probabilistic directed acyclic graph type with nodes as variables and the edges as their relationships. Based on the relationships, a node can "walk" to another. Each node has a probabilistic value and at the end of the walk, a final probabilistic score is formed. Relationship links that have high true positive score will be verified and formed into rules. Therefore, a Bayesian network is proactive even in misuse mode. However, its performance varies greatly. The reported precision rates are 89%, 99%, 21%, 7%, and 66% for DoS, probe/scan, remote-to-local, user-to-root, and "other" classes of attacks respectively.

Some popular clustering models are **k-means, k-nearest neighbor**, density-based spatial clustering of applications with noise (DBSCAN), etc. Because the models were designed in order to find patterns in unlabeled multi-dimensional data, explicit descriptions of classes are not required. A weakness of this model is known as the "curse of dimensionality". Too many features may confuse the model and any imbalance in the feature set will negatively affect its decisions.

Decision tree is a flow-chart like structure built on concepts of information gain/entropy where each node choose the best fit attribute to split current set of examples into subsets. Normally, decision trees provide the benefits of high accuracy with simple implementation. However, it is not usually the case with larger trees. Also with large, complex trees, the model tends to favor attributes with more levels. To overcome issues with large trees, analysts will have to do some pruning to get smaller trees.

While **Genetic Algorithm** (GA) and Genetic Programming (GP) are most used Evolutionary Computation (EC) methods; Particle swarm optimization, Ant Colony Optimization, Evolution Strategies are also parts of the group. The main concept is based on the idea of "the strongest will prevail" and basic operators are selection, crossover, and mutation. In misuse mode, an initial set of features and population will be picked and in the end, the best rules will surface to be used in a rule-based module. Experiments with various attack types show that the average false alarm rate is very low. However, the sensitivity in detecting new attacks varies greatly (from 66% to 100%) depending on attack types.

Naive Bayes model calculates the final conditional probability of "attack" (or "normal") with a strong (naive) assumption that the used features are independent from each other. That assumption is the biggest limitation of this model. However, if the features are indeed independent from each other, naive bayes can be very powerful thanks to its simple algorithm that allows the model to be highly scalable and be used as an online classifier.

Supported Vector Machine (SVM) is a binary classification model by design. With a kernel such as linear, polynomial, Gaussian Radial Basis Function, or hyperbolic tangen; the model will try to draw a hyperplane that divides the feature space into two classes. Sometimes, when overlapping is unavoidable, slack variables will be added and each overlapping data point will be be assigned a cost value. In misuse detection experiments, a large set of features is reduced by using feature selection policies or feature selection algorithms. The model is quite accurate but also shows limitations at identifying certain types of attacks such as user-to-root attack. In anomaly detection mode, usually SVMs will use more sophisticated kernel to help with the drawing of the hyperplane. Experimental results show great variations in accuracy (from 65% to 99.9%) and sometimes, false negative rate can get really high (over 30%)

2.2 Theoretical methods to attack ML models

In 2016, Tramer et. al. [24] proposed several methods to perform model extraction of several ML types. The strong possibility of a model being cloned gives our white-box attack a much better practical sense. In production environment, adversaries cannot freely probe the targeted model and risk being detected. They would have to find ways to clone the model(s), and only then they can perform off-line attacks. Because of this relationship, we believe it is very important to discuss Tramer's methods in the sections below.

2.2.1 Equation-solving attack. This form of attack is fit for logistic regression types such as binary logistic regression (BLR), multi-class logistic regression (MLR), and multi-layer perceptron (MLP). Because the models can be represented as equations with variables, attackers just need to feed the known variable values in, use mathematics to solve the equations and get the rest of the unknown values. For example, with BLR, we have :
$w \in R^d, \beta \in R$ with $f_i(x) = \sigma(w \times x + \beta)$
Attacker will feed x_i to the trained model and the model will give $y_i = f(x_i)$. If we have enough x_i, y_i, we should be able to solve the equations to get w and β. The methods proved to be effective in Tramer's experiments [24]. With BLR, they were able to achieve $R_{test} = R_{unif} = 0$ with an average probe of 41. The number of probe is much greater with MLR and MLP. For instance,it required them to perform 54,100 queries on average in order to achieve 100% accuracy on a 20 hidden node MLP. Unlike BLR, it is sometimes very hard to estimate a correct amount of probes needed for MLP and MLR cloning. Especially with MLP, it is hard for attackers to guess how many hidden neuron layers are there, and how many neurons per layer. Attackers will also not be able to tell how many classes an original MLP can identify. However, everything can be different in actual cyber attack scenario. Instead of 100% accuracy, attackers may only need to clone a model with 90% accuracy for their purposes, and the amount of probes needed may be significantly lower. Another reason to not aiming for 100% accuracy is that original ML models get tunned on a fast pace, daily. A 100% accurate cloned model of today maybe different from the actual model next week.

2.2.2 Model inversion attack. Given feature dimension d with feature vector $x_1, x_2, ..., x_d$, some knowledge about some of the features, and access to f - the model, Fredrikson et al. [11] proposed that a black box model inversion attack which involves finding an optimal x that maximizes the probability of some known values.
For instance, if an image of Bob was used to train model M to recognize category "man". If that exact image is fed into M, the result will be "man" with 100% confidence while images do not belong to the training set will never get such absolute score from the model. An attacker can start with one pixel and find the pixel value that gives the maximum score possible of category "man". The process continues to other pixels and the end result is an image very close to Bob's original image in the training set. Tramer et al. [24] upgraded this approach by performing inversion attack on a

cloned model M' of M. The reported improvement is a 6-hour faster recovering time for 40 faces. This kind of attack opens a theoretical possibility of which attackers can gain some insightful knowledge about a security model's trained data set if they could clone the model with 100% accuracy and somehow was able to tunnel it out.

2.2.3 Path-finding attack. Tramer also extended prior works on tree attacks and proposed their "path-finding" attack which can be used to map binary trees, multi-nary trees, and regression trees. We have a tree T with v nodes and at each node, there is an identifier id_v. With $x \in X$, an oracle query will give $O(x) = id_v$. If $x \in X_1 \cup \perp \times ... \times X_d \cup \perp$, $O(x)$ will return the identifier at the node where T stops. To begin the attack, we pick $x \in X_1 \cup \perp \times X_2 \cup \perp \times ...X_{i1} = [a, b] \times ... \times X_d \cup \perp$. $O(x)$ gives id_{Lv} at the leaves of the tree. We then can separate $[a, b]$ into n sub ranges where n is the number of the corresponding known leaves (at this point). For each X_{i1} sub range, we repeat the process and find another nodes/leaves. This was referred to as the top-down approach which is of higher performance than the bottom-up approach. Reported performance evaluations of this approach show that in order to achieve 100% on $1 - R_{test}$ and $1 - R_{unif}$, it will take 29,609 queries to clone a tree with 318 leaves, 8 layers of depth; 1,788 queries to clone a tree with 49 leaves, 11 layers of depth; and 7,390 queries to clone a tree with 155 leaves and 9 layers of depth.

2.2.4 Other attacks. There are several more ways to attack ML models. The Lowd-Meek attack [16] targets linear classifiers that give only class labels as models' outputs. The general idea of this approach is using adaptive queries to throw sample points at the suspected positions of the hyperplane. Another way to attack was described by Bruckner [5] as a single Stackelberg Prediction Game (SPG). In this game, the Leader (L) is the one with the original ML model M. the Follower (F) is the attacker. F will attack L by generating and feeding model M data that at least will prevent M from learning new knowledge or at most, teach M new faulty knowledge. Theoretically, this can be achieved by providing learning data that maximize the cost function of model M. In real life situations, there are more than one attacker with different attacking goals and L does not know how many F are there and what exactly each F is trying to do. This escalates to the Bayesian Stackelberg Game. Zhou and Kantarcioglu described it as "Nested Stackelberg Game" [28] suggesting a solution of using and switching a set of models to confuse the attacker. Kantarcioglu later on also proposed the concept of "planning many steps ahead" in this game. Details of these methods will be further studied and evaluated within a defense-in-depth environment and will be discussed in future works.

3 BACKGROUNDS ON STRATOSPHERE

In this section, we dive further into of Stratosphere and shed lights on its inner workings, the reasons behind the design decisions, on its beauty and weakness etc. The knowledge in this section will also provide insights into the reasons behind our chosen attack methods in the next main section.

Stratosphere has three main components: Argus [1], the ML Model Maker (a.k.a "Stratosphere Testing Framework") [3], and the Stratosphere itself [12]. The relationships between the network, the Stratosphere components and the users are illustrated in Figure 1.

3.1 The general structure

Argus is a commonly used network monitor tool that can either sniff live network traffics or parse entries recorded in pcap files. Argus correlates and groups network traffics into flows based on a tuple of Source IP/Port, Destination IP/Port, and Protocol. Based on these flows, an analyst will use the ML Model Maker to manually derive a unique behavior pattern for each type of malicious traffics. Such behavior pattern can be called a "model", be saved on disk and be loaded by the StratosphereIPS for live intrusion detection and prevention.

This design allows flexibility and safety in deploying machine learning models. We note that each model can be designed to deal with a specific type of problem - for example, a certain family of bot net. By limiting the scope of the model as such, we also limit the false negative rates. When the variations get out of the scope of a current model (the emergence of a new family of bot net built on top of the old one), a new model can always be created and deployed in parallel with the existing model. Models can also be version-ed and rolled back whenever necessary.

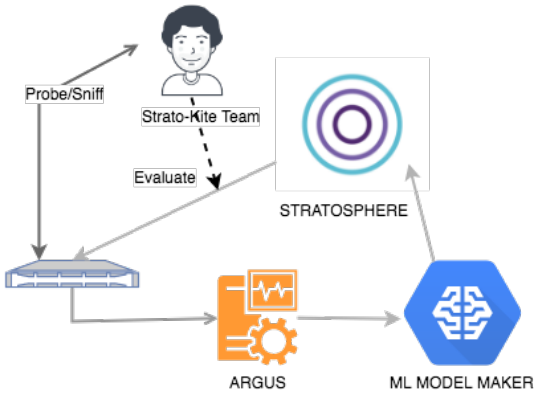

Figure 1: Probing Stratosphere

3.2 Core components of a model

Stratosphere's model is built based on four components: 1) The flows 2) The behavioral states 3) The Markov chains 4) The winner model.

As mentioned in the above sections, the flows were given by network tools like Argus [1]. Flows will then be categorized into 36 behavioral states based on flows' size, duration, periodicity and the time between consecutive flows. Each behavioral state will be given a predefined alpha-numeric character [12]. For example, "a" stands for a flow of short duration, small in size but strong in periodicity - a common character of DNS tunneling malware. A collection of behavioral states will form a Markov chain. Figure 2 shows a sample Markov chain of weather.

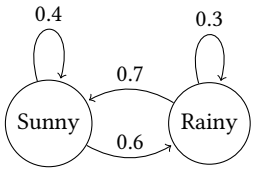

Figure 2: A Sample Markov Chain

Markov chain is Bayesian based of which definition and limitations were mentioned in Section 2.1. Imagine this Markov chain represents the Florida weather in 2017. For example, "0.4" is the probability that a sunny day will be followed by another sunny day. However, Florida weather in 2017 is not identical to 2016, 2015 and so on. We will then have multiple Markov chains representing Florida's weather. Fortunately, we can always find a generalized chain that can be used to identify Florida's weather from other States' weather no matter what year it is. In the context of Stratosphere, the States represent different family of malwares, the weather represents each malware's behaviors and the generalized Markov chain is called "the winning model".

In the following sections, we will discuss methods on how to attack winning models but since most Stratosphere's default models are about bot net identification, let's first take a look at how the detection of bot net works.

3.3 Detection of Botnets

Botnets are the technological backbone behind a myriad of attacks like identity stealing, DoS, SPAM, Advanced Persistent Threats(APTs) and government-sponsored attacks. A botnet is a network of remotely controlled compromised systems which is used for malicious activities. Hosts in a botnet are called as bots and the owner of a botnet as Bot-master. Initially, Botnets were implemented with IRC protocol which is a centralized model and later moved to P2P protocol and finally shifted to HTTP protocol [9]. So, there are many different perspectives in detecting botnets, mainly host-based and network-based detections. In host-based systems, we have traditional antivirus, antimalware, while network-based techniques we have IDS, IPS which focus on packets in the network. Initially, botnets, where detected with static analysis techniques, as the complexity of botnets infrastructure, is increasing, researchers are working hard on creating algorithms which can successfully detect botnet traffic. The advances in machine learning and access to better botnet traffic datasets start showing promising results with the help of behavioral models. Botnets are changing continuously because of two things namely DGA (Domain Generation Algorithm) which generates a new domain name from time to time and next is the Fast-Flux technique which basically changes IP address for a domain very quick [26].

Botnets were initially detected using honeypots and passive traffic analysis [27]. The complexity of botnets has increased; hence detection of botnets is done using network-based techniques[20]. These network-based techniques can be classified into four categories based on Detection sources, Detection features, Detection techniques, Detection algorithms [27].

Detection Sources: This category deals with the information source used by the network-based models in detection. These can be network packets from honeypots, net-flows, application logs.

Detection Features: This is related in identifying what features are being used in previous research for successful detection of botnets, like encrypted botnet detection, botnet protocols.

Detection Techniques: It is related with what type of techniques are used. For example: behavior, static, fingerprint, anomalies or signature-based techniques.

Detection Algorithms: In this category, the detection techniques are further classified into supervised, semi-supervised, unsupervised, signal processing and Heuristic rules.This classification is elaborately depicted in the figure below. [21]

Table 1 summarized common bot detection techniques on which we elaborate more in the below paragraphs.

With BotSniffer [15], detection involves sniffing network traffic and grouping hosts connecting to the same remote server and port. One host that triggered the attack will make the entire group of such hosts marked a possible botnet. Detection is also based on IRC protocol responses from the hosts. It uses the DICE coefficient to create clusters. Sequential probability ratio testing will then determine whether a group is really a botnet or not. On top of sequential probability ratio testing, the paper applies threshold random walk (TRW) to further decide if there is a botnet infection.

BotMiner [2] uses the traffic flow information and two-step X-means clustering methods to create groups. Groups are created based on the similarity of the attack type such as SPAM, exploit attempts, port scans, and binary downloads. It then checks the similarity among individual hosts to identify a potential botnet by creating a dendrogram and finding out the best cluster in it.

The BotHunter framework [14] makes use of a state-based infection sequence model to detect infected hosts and their botnet communication. This proposal uses a modified Snort IDS which is fed all the inbound and outbound network packets. It performs bot dialogue correlation analysis and uses the IDS warnings to modify the score of each host in the network. A host is said to be infected if an evidence of outward bot coordination or local host infection is found and at least two distinct signs of outward bot coordination or attack propagation are found.

N-gram [17] proposes an unsupervised, classification-based and IRC-based botnet traffic detection method. First classifies the application traffics and then detects bots. Network traffics will first be categorized into known and unknown applications with the help of known ports and signature-based detection. A decision tree is then used to classify the unknown application traffic based on temporal features. Feature analysis will be performed on the unknown applications traffics using K-means, unmerged X-means and merged X-means algorithms. Finally, the cluster with least standard deviation is labeled as botnet.

Unclean paper [8] leverages historic network logs to predict future hostile activities using spatial uncleanliness and temporal uncleanliness properties. A dataset consisting of external reports of phishing, SPAM activity, and port scans, and internal network captures is used to evaluate both the uncleanliness properties. The paper compares the population of IP addresses in an unclean report

Papers	Bot behavior	Botnet behavior	Temporal behavior	Protocol behavior
BotMiner [2]	Pattern of packets per flow and mean bytes per packet ; connects to many SMTP servers; ask many MX records	Same attack is ordered to every bot; synchronized attacks; bots have similar traffic patterns	Mean bytes per second and flows per hour	Port scan detection; many MX records asked; packet and byte transfer; fph, ppf, bpp and bps
BotSniffer [15]	Binary download?; send SPAM?	Attacks are synchronized; IRC answer messages are synchronized	Attacks are in the same time windows; IRC responds to PRIVMSG in the same time window	Port scan detection
BotHunter [14]	Attack signature prior knowledge	None	Time windows are used	Port scan detection
Appendix B of [15]	Connects to the C&C	None	Bot connects to HTTP C&C periodically	None
N-gram [17]	None	None	Features are computed within a time interval	Features are computed for each protocol
Unclean [8]	SPAM and phishing	Botnets infect the same unclean networks	Bots appear in the same networks over time	Port scan detection
FluXOR [19]	None	Anomalous changes in the features of domains	None	None
Tight [23]	None	Botnet controls bots at the same time	C&C server sends orders at the same time	bpp, bps and pps

Table 1: Comparison of anomaly-based behavior detection techniques

against normal population of IP addresses in the Internet for evaluating the spatial uncleanliness property. Whereas for evaluating the temporal uncleanliness property the paper tests the unclean network activity using an old report of unclean addresses to predict compromised IP addresses. To identify innocent, unknown and hostile IP addresses from the possible unclean bots in the network the paper analyzes the performance impact of blocking these possible hosts in the second phase.

FluXOR [19] tries to detect the fast-flux enabled domains and the agents who are involved in a fast-flux service network. The domains information collected by the collector module is been monitored by the monitor module for changes in the domain WHOIS and DNS information. The output is fed to the classifier algorithm from the Waikato Environment for Knowledge Analysis suite.

So as the most of the above detection methods claim to detect botnets with very less false positives. However, these models fail when they detect unknown botnets [22], BClus, CCDetector. These algorithms work on basis of flows [13] Markov chain [4] instead of network packets. A flow is a conceptual structure that aggregates and summarizes all the packets sharing the following five fields: source IP, source port, destination IP address, destination port and protocol. The Stratosphere team has also compared their efficiency of detection using the malware capture dataset from the Malware Capture Facility Project[MCFP] also compared efficiency with CAMNEP method, a commercialized method to detect botnets.

4 ETHICAL GUIDELINES

1. We completely isolate our network test environment from others by preventing access to all physical and virtual network interfaces. Our methodologies rely on the manipulating and parsing of network capture files that were properly prepared by other institutions. Access to those files are limited within the Stratokite team with expected zero leak.

2. We employ strict use of virtual machines (VMs) or air-gap computers. That means all potential harms should be limited within the scope of the VMs and air-gap computers. All necessary softwares will be downloaded and shared in read-only mode with the VMs.

3. We follow strict use of legal tools. All tools used are legal and recommended by researchers and security professionals. We do not use malware/virus source codes for testing our models.

4. We do not modify the software, trained models, or other components came in default configurations of the StratosphereIPS public release.

5. We will keep all major security vulnerabilities confidential and use proper report channels to notify the Stratosphere team before publishing.

5 ATTACKING STRATOSPHERE

Before we discussing our methodology of attacking Stratosphere, it is important to really define the meaning of "attacking" a machine learning model in general. Because a ML model is a moving target, constantly tuning itself to adapt to the changes in the environment, the strategy of attacking it should be flexible and be focusing on the models' purposes rather than the models' functionalities.

The accuracy rate is not the only thing adversaries can target. In one case, if a model was designed to allow or drop suspicious packets then reducing the model's accuracy fits the definition of an attack. In another case, adversaries can cause the models to produce true positives that are very close to false positives. Consequently, it causes burn-outs on the security analysts who are going to manually inspect those flags. The model's functionalities are intact but one of its purposes - reducing the security analysts' workloads - was compromised. "Attack" can also mean significantly increasing the time it takes for a ML model to make a specific decision ("untrain the model").

No matter what the attack end-goals are, it is not really an attack on a particular model if the adversaries did not extract (clone) the model. Metaphorically, getting through an opening/closing trap door one time does not mean anything if the open-close rhythms change constantly. Only when we know how new rhythms are generated then we can say we "owned" the trap door. Due to the limited scope and time constraint, in this version of the paper, we will only focus on the "one time" hacking methods and cannot stress enough that these methods are only meaningful when deploying in the first stage of the ML Kill Chain [18].

5.1 Core Methodology

Our methodology is white box based and in the specific case of StratosphereIPS, we abuse the detection score, which is also a common feature in other products. Since each model in Stratosphere was made to deal with a very narrow set of malicious activities, excessive mutations of original payloads will obviously defeat a particular model and give no practical meaning (since such mutations may be detected by other models loaded into the program). Therefore, the main goal should be about making as few changes as possible to the malicious behavior patterns while evading detection (getting detection score as close to zero as possible).

Execution involves two main functions: Explore and Blending. Explore function will take a stream of malicious payloads and try to identify which part of the behavioral signature contributes most to the detection score. Such part is called "hot spot".

Algorithm 1 Explore algorithm

```
1:  m ←malicious-payloads                    ▷ locate pure malicious contents
2:  score ←detect(m)                         ▷ get detection score of m
3:  M[n] ←chop(m)                            ▷ chop m into n sections
4:  procedure EXPLORE(M)
5:      hotspot ←0
6:      i ←0
7:      for i < n do
8:          s1 ←detect(M[i])
9:          s2 ←detect(M - M[i])             ▷ M without M[i]
10:         if max(s1, s2) > score then
11:             score ←max(s1, s2)
12:             hotspot ←i
13:     return timeframe(hotspot)            ▷ Return start/end time
```

A hot spot is then be reported to the Blending function which will blend normal traffics into the spot. Normal traffics can be non-malicious packets from a non-malicious dataset or no packet at all. This will virtually affect the size, periodicity, and timing of the flows; causing noises in the behavioral signature. The functions can be looped until a certain low detection score can be met.

Algorithm 2 Blending algorithm

```
1:  m ←malicious-payloads                    ▷ pure malicious contents
2:  n ←normal-payloads                       ▷ non-malicious contents
3:  procedure BLEND(n,m)
4:      frame[start, end] ←Explore(m)
5:      M[3] ←chop(m, frame[start], frame[end])
6:      N ←extract(n)                        ▷ take a portion of n
7:      M[2] ←interlace(M[2],N)
8:      export ←(M[1] + M[2] + M[3])
9:      export ←timeSync(export)
10:     return export
```

It is important to note that we do not explicitly specify the number of chopped blocks (the n number) when exploring a malicious payload capture, nor how big is the normal data portion to be blended into the hot spot. If the number of chopped blocks is too high, many loops may be needed and it may become inefficient. We recommend the measuring unit to be minutes and the time duration of the normal data portion picked should not be greater than the time duration of one chopped block. For example, if we want to explore a 15 minutes window of malicious traffics and we chop it into 3 blocks, the normal data block to be blended in to one of the three malicious blocks should not have the time duration greater than 5 minutes.

5.2 Tools

In this section we describe the various tools we are planning to use in probing the machine learning models of the Stratosphere IPS, namely Argus and WireEdit.

Argus: It is a network Audit Record Generation and Utilization System. The main focus of Argus is to develop all aspects of large scale network situational awareness derived from network activity audit. Argus, itself, is next-generation network flow technology, processing packets, either on the wire or in captures, into advanced network flow data. For our experiments, we are going to use Argus for correlating packets into flows. With help of the flows from the malicious captures and loading different models in the Stratosphere we can find out the particular model triggered in detection of the malware capture. This will help in narrowing the flow and the model responsible for detection of malware captures by the Stratosphere IPS.

Argus captures most of the packet dynamics and semantics of each flow, with a great deal of data reduction, which helps in storing, processing, inspecting and analyzing large amounts of network data efficiently. Argus provides reachability, availability, connectivity, load, retransmission, and delay metrics for all network flows, and captures most attributes that are available from the packet contents, such as L2 addresses, tunnel identifiers (MPLS, GRE, ESP) protocol ids, hop-count, options, L4 transport identification (RTP, RTCP detection), host flow control indications.

WireEdit: After narrowing down a particular flow and model responsible for successful detection of malware capture by the StratosphereIPS, we try to manipulate the packet responsible for the flow using WireEdit in such a way the model of Stratosphere is unable to detect this malicious packet. We do not try to change the payload of the packet and hence the packet is still malicious. This tool is so powerful as it provides flexibility in modifying the packet at any stack layer from L2 to L7 with just a few keystrokes. It is also easy to use, as you don't have to know the encoding rules and syntax before editing the packet captures.WireEdit verifies all the changes you make on-the-fly and takes care of fields/layers encoding, offsets, inter-dependency, integrity and checksum.

5.3 Preparation

In this section we describe how different pcap's were generated. We generated different sets of pcap's by following the above mentioned algorithms in section 5. The Stratosphere IPS concentrates on flows and each flow has 4 factors associated with it. A flow is described as collection of packets which have same source IP, same destination IP, same destination port and same protocol. The 4 factors associated with a flow are: Size of the flow, duration of the flow, periodicity of the flow, time between consecutive flows. So we prepared our dataset based on these factors. We used the malware capture-CTU-Malware-Botnet-25-3 which is a zeus malware pcap from the Malware Capture Facility Project. This malware capture is captured from an Zeus infected machine for 4 hours. With this pcap as the baseline we try to generate various combinations. In this malicious capture we concentrate on the first 15 minutes as the entire pcap is malicious and contains only DNS requests and DNS responses. Our assumption is that the zeus malware is performing DNS tunneling to connect and exfiltrate data to a Command and Control server. So from Table 2 you can see that different packet captures are generated from the Zeus packet capture based on factors that Stratosphere IPS use in detecting malicious behavior. In table 2, type of packets represent the contents of packet capture: ALL represent the capture contains all type of packets (tcp, udp, ARP, DNS). On the other hand type DNS denote that the capture only has DNS packets. Also the default time window used by the Stratosphere IPS in monitoring network traffic is 5 minutes. Hence we sliced the captures to 5 minutes intervals denoting time frame of that slot, for example the first tuple consists of All type of packets and malware packets between minutes 0 to 5 in slot 1 , malware packets between 5-10 minutes in slot 2 and malware packets between 10 -15 minutes in slot3.

Similarly, we also generated more packet captures, this time added normal captures from the malware capacity facility project to the Zeus malware capture. From table 3 we can see different packet captures generated. A tuple from table 3 can be described as follows: If we consider the first tuple, ALL in type of packets field represents that the capture contains all type of packets(tcp,udp.ARP,DNS) and D5-1 means first five minutes of normal traffic in slot 1, M5-1 means first five minutes of zeus malware packets in slot 2, D5-2 means second five minutes of normal packets in slot 3, M5-2 means second five minutes of zeus malware packets in slot 4, D5-3 means third five minutes of normal packets in slot 5, M5-3 means third five minutes of zeus malware packets in slot 6 and D5-4 means fourth five minutes of normal packets in slot 7.

```
genterists-Mac-mini:StratosphereLinuxIPS genterist$ ra -F ./ra.conf -n -Z b -r ~/Downloads/dryrun.argus | p
ython2.7 ./slips.py -f ./models -d -v 3
Stratosphere Linux IPS. Version 0.3.4
https://stratosphereips.org

Detecting malicious behaviors with the following models:
        Adding model From-Botnet-TCP-HTTP-CC-23 to the list.
        Adding model From-Botnet-TCP-HTTP-CC-25 to the list.
        Adding model From-Botnet-TCP-HTTP-CC.BitcoinMiner-1 to the list.
        Adding model From-Botnet-TCP-HTTP-CC.Cridex-1 to the list.
        Adding model From-Botnet-TCP-HTTP-CC.Miuref.Attempt-4 to the list.
        Adding model From-Botnet-TCP-HTTP-CC.PlainText-1 to the list.
        Adding model From-Botnet-TCP-HTTP-Flu.CC-3 to the list.
        Adding model From-Botnet-TCP-HTTP-Zeus.CC-1 to the list.
        Adding model From-Botnet-TCP-Unknown-CC-1 to the list.
        Adding model From-Botnet-TCP-Unknown-CC.Attempt-2 to the list.
        Adding model From-Botnet-TCP-Unknown-Linux.Botnet-1 to the list.
        Adding model From-Botnet-TCP-Unknown-Lurk.CC-1 to the list.
        Adding model From-Botnet-UDP-DNS-DGA-1 to the list.
        Adding model From-Botnet-UDP-DNS-DGA-17 to the list.
        Adding model From-Botnet-UDP-DNS-DGA-18 to the list.
        Adding model From-Normal-TCP-HTTP-LastFM-30 to the list.
Time Window Started: 1961-06-29 22:35:43, finished: 1961-06-29 22:40:43. (1 connections)
Time Window Started: 1969-12-31 19:00:46.958782, finished: 1969-12-31 19:05:46.958782. (36 connections)
Time Window Started: 1969-12-31 19:05:46.961512, finished: 1969-12-31 19:10:46.961512. (4 connections)
        +10.0.2.103 verdict: Malicious (SDW score: 0.00361) | TW weighted score: 0.0357142857143 = 1.0 x 0.
0357142857143
                10.0.2.103-8.8.8.8-53-udp [Level 3 Communications, Inc.,US] (198/198)
                        DstIP: 8.8.8.8, Label:             From-Botnet-UDP-DNS-DGA-17 , Detection Time:196
9-12-31 19:05:46.961512, State(100 max): 44.R+R.R.r.a.a.d.a.a.a.a.a.a.a.a.a.a.a.a.d.a.a.a.d.a.a.a.a.a.a.a.d.d
.b.D.D.a.a.a.a.d.a.a.a.a.a.a.d.d.a.a
```

Figure 3: Dry Run Result

5.3.1 Dry run. The Stratosphere IPS provides a feature of running a packet capture through it to scan and detect malicious botnet activities with the use of various available Markov Chain based Machine Learning models. In this section we run the Zeus botnet packet capture through the Stratosphere IPS to examine and analyze its output.

(1) Malicious packet capture: The Zeus botnet malware capture is made publicly available by the Stratosphere team [12]. They provide the Zeus packet captures for 4 hours, 3 days, 6 days, 34 days, 40 days and 4 months. We will be focusing first on the 4 hours dataset and manipulating the corresponding packet capture.
(2) Argus and network flows: The Stratosphere is built on top of the network activity auditing tool: Argus. In that it uses Argus to identify different network flows and use it to efficiently detect malicious botnet activities within the network.
(3) Stratosphere IPS and malicious flows: The Stratosphere IPS makes use Markov Chain based Machine Learning models to identify malicious botnet activities within a network. As of April 13, 2018 the Stratosphere team has made sixteen models, including the Zeus model. On performing a dry run for all kinds of Zeus botnet packet capture the IPS actually did detect the malicious flow in the malware capture as seen in the figure 3.

5.3.2 Narrow down. We narrow down and identify the botnet flows within the network captures that was detected as suspicious/malicious by the IPS. This helps us in identifying the exact botnet flow which is to be manipulated such that we can change the behaviors of those flows in order to create more true negatives and false positives using the tools mentioned in Section 5.2.

6 RESULTS

In this section we present you the results of our experiments performed on the Stratosphere IPS based on the dataset generated in section 5.3. With the Pcap's generated we first generate the argus

S.no	Type of packets	Slot 1	Slot 2	Slot 3	Factor of Stratosphere
1	ALL	00-05	05-10	10-15	Baseline
2	ALL	-	05-10	10-15	Duration
3	ALL	00-05	-	10-15	Duration
4	ALL	00-05	05-10	-	Duration
5	ALL	05-10	10-15	00-05	Periodicity
6	ALL	05-10	00-05	10-15	Periodicity
7	ALL	10-15	00-05	05-10	Periodicity
8	ALL	10-15	05-10	00-05	Periodicity
9	DNS	00-05	05-10	10-15	Size(Reduction)
10	DNS	-	05-10	10-15	Size(Reduction) and Duration
11	DNS	00-05	-	10-15	Size(Reduction and Duration
12	DNS	00-05	05-10	-	Size(Reduction) and Duration
13	DNS	05-10	10-15	00-05	Size(Reduction) and Periodicity
14	DNS	05-10	00-05	10-15	Size(Reduction) and Periodicity
15	DNS	10-15	00-05	05-10	Size(Reduction) and Periodicity
16	DNS	10-15	05-10	00-05	Size(Reduction) and Periodicity

Table 2: Generation of Dataset from the zeus malware capture

S.no	Type of packets	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7	Slot 8
1	ALL	D5-1	M5-1	D5-2	M5-2	D5-3	M5-3	D5-4	-
2	ALL	D5-1	D5-2	M5-1	D5-3	M5-2	D5-4	M5-3	-
3	ALL	D5-1	D5-2	D3-1	M5-1	D5-3	M5-2	D5-4	M5-3
4	ALL	D3-1	M5-1	D3-2	M5-2	D3-3	M5-3	D3-4	-

Table 3: Generation of Dataset from the Zeus malware and the Normal capture

files for them using the argus tool and finally provide these argus files to the Stratosphere IPS as input. We do not modify any models from the Stratosphere IPS. From table 4 we can see that by shifting packets across various time slots, the Stratosphere IPS still successfully detects the malicious packet captures. The Stratosphere IPS fails to detect only in 3rd, 4th and 11th tuples in the table 4. Hence we do not consider those packet captures for the next set of experiments as Stratosphere fails to detect it. This table conveys that even though we shift packets across various time intervals it is still considered to be malicious as most of the malware captures are UDP based and DNS requests and responses. Hence we do not break any connections and the packet captures generated are still malicious.

Now in the next set of experiments we use the dataset generated from table 3 in section 5.3. The results of these are presented in table 5 and we can see that the Stratosphere IPS fails to detect any malicious behavior in these packet captures. As we just add normal packets in different time slots and do not change anything in the malicious packets, the packets are still considered to be malicious. The first tuple in table 5 is similar to first tuple in table 4 but with an addition of normal packets in various slots. The Stratosphere IPS detected 2 malicious detections for the first tuple in table 4, however failed to detect anything in first tuple of table 5. Similarly, the Stratosphere IPS failed to detect anything for tuples 2, 3, 4 and 5 of table 5. In Figure 4 we probe the Stratosphere IPS with time difference between two consecutive flows. In Figure 5 we probe the Stratosphere IPS with Mixed captures-Normal and malicious packets within the same time slot and in figure 6 we probe the Stratosphere IPS with Mixed Captures-Normal and malicious packets across different time slots. The results are shown below in figure 4, figure 5 and figure 6 and they depict that Stratosphere fail to detect malicious behavior from the mixed packet captures. In the figures 4, 5, 6 provided below we can see that we haven't manipulated any models of the Stratosphere IPS and the time window for Stratosphere IPS is 5 minutes . We first generate the argus files for all the packet captures and provide it to the Stratosphere IPS. Also we can see that Stratosphere IPS detects 0 malicious IP from the figures below.

7 RECOMMENDATIONS

From the beginning, ML scientists should pay attention to the attack model and at least develop a list of recommendations for safe implementations. Recommendations may include but are not limited to the designer's definition of "attack", the meaning of model's accuracy, the side channels, etc. This is essentially important in the context of open-source. As mentioned before, the definition of "attack" should depend on the model's intended purposes rather than just its accuracy. Some ML based solutions were designed to be multi-purposes. Some solutions were originally designed for a

S.no	Type of packets	Slot 1	Slot 2	Slot 3	Factor of Stratosphere	No.of Malicious Detections
1	ALL	00-05	05-10	10-15	Baseline	2
2	ALL	-	05-10	10-15	Duration	1
3	ALL	00-05	-	10-15	Duration	0
4	ALL	00-05	05-10	-	Duration	0
5	ALL	05-10	10-15	00-05	Periodicity	3
6	ALL	05-10	00-05	10-15	Periodicity	3
7	ALL	10-15	00-05	05-10	Periodicity	3
8	ALL	10-15	05-10	00-05	Periodicity	3
9	DNS	00-05	05-10	10-15	Size(Reduction)	3
10	DNS	-	05-10	10-15	Size(Reduction) and Duration	1
11	DNS	00-05	-	10-15	Size(Reduction and Duration	0
12	DNS	00-05	05-10	-	Size(Reduction) and Duration	1
13	DNS	05-10	10-15	00-05	Size(Reduction) and Periodicity	3
14	DNS	05-10	00-05	10-15	Size(Reduction) and Periodicity	3
15	DNS	10-15	00-05	05-10	Size(Reduction) and Periodicity	3
16	DNS	10-15	05-10	00-05	Size(Reduction) and Periodicity	3

Table 4: Results from Stratosphere IPS with Dataset from table 2

S.no	Type of packets	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7	Slot 8	No. of Malicious Detections
1	ALL	D5-1	M5-1	D5-2	M5-2	D5-3	M5-3	D5-4	-	0
2	ALL	D5-1	D5-2	M5-1	D5-3	M5-2	D5-4	M5-3	-	0
3	ALL	D5-1	D5-2	D3-1	M5-1	D5-3	M5-2	D5-4	M5-3	0
4	ALL	D3-1	M5-1	D3-2	M5-2	D3-3	M5-3	D3-4	-	0
5	ALL	D3-1	M2-1	D2-1	M3-1	D1-2	M3-2	D4-2	-	0

Table 5: Results from Stratosphere IPS with Dataset from table 3

Figure 4: Probing Stratosphere: Time between consecutive flows

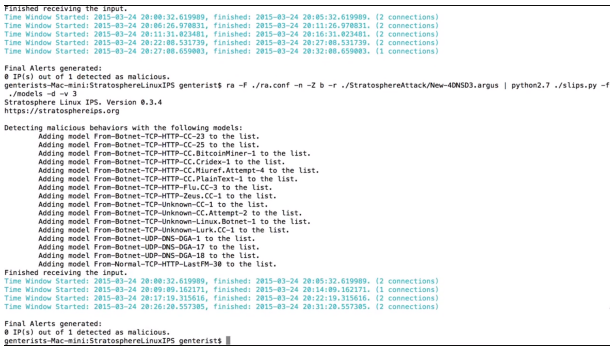

Figure 5: Probing Stratosphere: Mixed captures within the same time slot

specific purpose but were used for other purposes in real-world implementations.

For example, most recent works in ML based security for SDNs have accuracy rates of 98% but the meaning of 0.2% increase in false negatives may differ greatly from one implementation to another. Based on the attack model, the ML designers may also provide a default protection model, explaining how the structure of their designs fit into the protection model, what may be done to harden their works, what are the security trade-offs to be considered, what are the potential side channels in real world deployments, and so on.

ML based solutions should also generate meaningful logs or even better, having an interface for the model to be audited automatically. Audits may include information on who made what changes, how much the model has drifted after a period of time, the rates of false

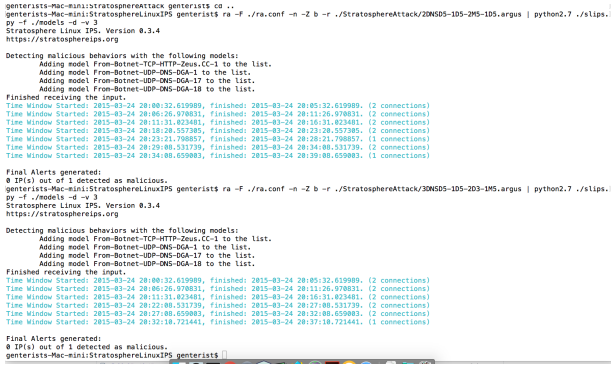

Figure 6: Probing Stratosphere: Mixed Captures across various time slots

positives and false negatives, etc. ML based solution with good audit capability will also help in case the model needs to be rolled back to its earlier versions.

Cost is another factor as important as accuracy. For the same purpose, a leaner ML algorithm will usually cost less than a complicated one but there may be cases where it is justifiable to have a complex ML model or even a group of different ML models working together. The designers should at least provide a cost model to make practical sense out of their design decisions. The paper "Machine Learning with Operational Costs" from MIT researchers [25] may serve as a good start for further readings into optimizing ML operational costs.

8 CONCLUSION AND FUTURE WORKS

Machine learning models is flexible and is changing all the times due to the moving landscape of attacks and defense. Successful evasions of the models do not guarantee persistent evasions for future exploits. Cloning the defending model is the best option for maintaining the "persistent" factor. Successful cloning process will not only copy classification results but also capture the training and drifting patterns of the models. This is extremely important in real-world environments where a model has to deal with multiple threat actors with different attack patterns and be tunned frequently.

Within our knowledge, we believe that using artificial neural network (ANN) will be good enough for a model clone. With Stratosphere, while each model is based on Markov Chain model, actual deployment involves multiple models being loaded at the same time. With possible models' overlapping and models' parallel processing of patterns, using math based attack methods (such as Equation attack) on Markov Chain may give incorrect results. ANN, on the other hand, will treat multiple Markov Chain models as one model and try to mimic its classifications.

For future works, we will develop a ML attack process with 3 major steps: Train, Test, and Predict.We used prepared datasets provided by team Stratosphere including: malicious-only dataset (MD), non-malicious-only dataset (ND), and mixed dataset (MixD). For training and testing, we will use MD and ND with 7-3 rule. That means 70% of each data set will be used for training and 30% will be used for testing. In the end, we will use MixD to predict performance of the defending model.

For training the clone model, we first run 70% of MD and ND through the real model (Stratosphere), observe and record the classification results. For each flow Fi, there is a corresponding classification Ci. We then feed the pairs into the clone model which is ANN based. Note that we do not record the confidence store provided by Stratosphere because it was provided by a single Stratosphere model while we are trying to model the collective behavior of all models loaded into Stratosphere.

For testing, we will feed each 30% of MD and ND into the clone model first and observe classification results together with associated confidence scores. We then run the same data through the defending model and observe classification results. At the end of the process, we should be able to compute the performance offset between the clone model and Stratosphere.

Finally, we will use the mix dataset to predict the behaviors of Stratosphere. We run any random flow in the dataset through the clone model, observe results, apply offset and calculate the final predicted result. It is important to note that we do not know for sure if the flow is benign or malicious. What we have is just the clone model telling us if the defending model will classify the flow as malicious or not. We then run the same flow through Stratosphere, observe and compare results. The clone model should be able to predict Stratosphere's classification with reasonable precision.

Successful clone of Stratosphere will allow attackers to perform off-line attacks. At the end of the process, hackers will be able to craft stealthy payloads with high confidence that the defending model will be unable to detect.

With the experiments and results shown in our paper we conclude that the machine learning models of the Stratosphere IPS are not accurate enough. We are able to bypass the Machine Learning models for the given time slots without triggering any detection for the malicious activity. Hence, a bot can perform malicious activity and ex-filtrate data to its C&C without being detected.

REFERENCES

[1] [n. d.]. ARGUS- Auditing Network Activity. ([n. d.]). https://qosient.com/argus/index.shtml

[2] [n. d.]. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. ([n. d.]). https://www.usenix.org/legacy/event/sec08/tech/full_papers/gu/gu_html/index.html

[3] [n. d.]. Stratosphere Testing Framework àĂŤ Stratosphere IPS. ([n. d.]). https://www.stratosphereips.org/stratosphere-testing-framework/

[4] Michel Bailey, Evan Cooke, Farnam Jahanian, Yunjing Xu, and Manish Karir. 2009. Al survey of botnet technology and defenses. In *Proceedings - Cybersecurity Applications and Technology Conference for Homeland Security, CATCH 2009*. https://doi.org/10.1109/CATCH.2009.40

[5] Michael Brückner and Tobias Scheffer. [n. d.]. Stackelberg Games for Adversarial Prediction Problems. ([n. d.]).

[6] Anna L. Buczak and Erhan Guven. 2016. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials* 18, 2 (22 2016), 1153–1176. https://doi.org/10.1109/COMST.2015.2494502

[7] Bureau of Labor Statistics. 2015. US Bureau of Labor Statistics, Occupational Outlook Handbook. (2015). http://www.bls.gov/ooh/transportation-and-material-moving/taxi-drivers-and-chauffeurs.htm

[8] M. Patrick Collins, Timothy J. Shimeall, Sidney Faber, Jeff Janies, Rhiannon Weaver, Markus De Shon, and Joseph Kadane. 2007. Using uncleanliness to predict future botnet addresses. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement - IMC '07*. ACM Press, New York, New York, USA, 93. https://doi.org/10.1145/1298306.1298319

[9] Richard Ford Director and Sarah Gordon. [n. d.]. Cent, Five Cent, Ten Cent, Dollar: Hitting Botnets Where it Really Hurts. ([n. d.]). http://delivery.acm.org.prox.lib.ncsu.edu/10.1145/1280000/1278942/p3-ford.pdf?ip=152.14.136.96&id=1278942&acc=ACTIVE%20SERVICE&key=6ABC8B4C00F6EE47%

2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1522290207_3156bc6758774377255ecdb0b300409a#URLTOKEN#

[10] Cisco V N I Forecast. 2017. Cisco visual networking index: Global mobile data traffic forecast update 2016-2021. *Cisco Public Information, February* (2017). https://goo.gl/tIlk4D

[11] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. [n. d.]. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. ([n. d.]). https://doi.org/10.1145/2810103.2813677

[12] Sebastian Garcia. [n. d.]. Stratosphere IPS for Linux. ([n. d.]). https://github.com/stratosphereips/StratosphereLinuxIps

[13] S García, M Grill, J Stiborek, and A Zunino. 2014. An empirical comparison of botnet detection methods. *Computers & Security* 45 (2014), 100–123. https://doi.org/10.1016/j.cose.2014.05.011

[14] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. [n. d.]. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. ([n. d.]). https://www.usenix.org/legacy/events/sec07/tech/full_papers/gu/gu.pdf

[15] Guofei Gu, Junjie Zhang, and Wenke Lee. [n. d.]. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. ([n. d.]). http://faculty.cs.tamu.edu/guofei/paper/Gu_NDSS08_botSniffer.pdf

[16] Daniel Lowd and Christopher Meek. [n. d.]. Adversarial Learning. ([n. d.]). http://dl.acm.org/citation.cfm?id=1081950

[17] Wei Lu, Goaletsa Rammidi, and Ali A. Ghorbani. 2011. Clustering botnet communication traffic based on n-gram feature selection. *Computer Communications* 34, 3 (3 2011), 502–514. https://doi.org/10.1016/j.comcom.2010.04.007

[18] Tam N. Nguyen. 2017. Attacking Machine Learning models as part of a cyber kill chain. (5 2017). http://arxiv.org/abs/1705.00564

[19] Emanuele Passerini, Roberto Paleari, Lorenzo Martignoni, and Danilo Bruschi. 2008. FluXOR: Detecting and Monitoring Fast-Flux Service Networks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer Berlin Heidelberg, Berlin, Heidelberg, 186–206. https://doi.org/10.1007/978-3-540-70542-0{_}10

[20] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sa, David Zhao, Wei Lu, and John Felix. 2011. Detecting P2P Botnets through Network Behavior Analysis and Machine Learning RCMP Atlantic Region. (2011).

[21] Sebastián García. [n. d.]. DetectingBotnetsbyModelingtheirNetworkBehaviors. ([n. d.]).

[22] W. Strayer, Robert Walsh, Carl Livadas, and David Lapsley. 2006. Detecting Botnets with Tight Command and Control. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. IEEE, 195–202. https://doi.org/10.1109/LCN.2006.322100

[23] W. Timothy Strayer, Robert Walsh, Carl Livadas, and David Lapsley. 2006. Detecting botnets with tight command and control. In *Proceedings - Conference on Local Computer Networks, LCN*. https://doi.org/10.1109/LCN.2006.322100

[24] Florian Tramèr, Fan Zhang, Floriantra Er Epfl, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs. In *Proceedings of the 25th USENIX Security Symposium*.

[25] Theja Tulabandhula and Cynthia Rudin. 2013. Machine Learning with Operational Costs. *Journal of Machine Learning Research* 14 (2013), 1989–2028. http://www.jmlr.org/papers/volume14/tulabandhula13a/tulabandhula13a.pdf

[26] Hossein Rouhani Zeidanloo and Azizah Bt Abdul Manaf. 2010. Botnet Detection by Monitoring Similar Communication Patterns. (4 2010). http://arxiv.org/abs/1004.1232

[27] Zhu Zhaosheng, Judy Fu Zhi, Lu Guohan, Roberts Phil, Chen Yan, and Han Keesook. 2008. Botnet research survey. In *Proceedings - International Computer Software and Applications Conference*. https://doi.org/10.1109/COMPSAC.2008.205

[28] Yan Zhou and Murat Kantarcioglu. [n. d.]. Modeling Adversarial Learning as Nested Stackelberg Games. ([n. d.]). https://doi.org/10.1007/978-3-319-31750-2