Detecting Bots using Stream-based System with Data Synthesis

Tianrui Hu

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Gang Wang, Co-chair
Bimal Viswanath, Co-chair
Bert Huang

April 29, 2020
Blacksburg, Virginia

Keywords: Bot Detection, Security, Machine Learning

Detecting Bots using Stream-based System with Data Synthesis

Tianrui Hu

(ABSTRACT)

Machine learning has shown great success in building security applications including bot detection. However, many machine learning models are difficult to deploy since model training requires the continuous supply of representative labeled data, which are expensive and time-consuming to obtain in practice. In this thesis, we build a bot detection system with a data synthesis method to explore detecting bots with limited data to address this problem. We collected the network traffic from 3 online services in three different months within a year (23 million network requests). We develop a novel stream-based feature encoding scheme to support our model to perform real-time bot detection on anonymized network data. We propose a data synthesis method to synthesize unseen (or future) bot behavior distributions to enable our system to detect bots with extremely limited labeled data. The synthesis method is distribution-aware, using two different generators in a Generative Adversarial Network to synthesize data for the clustered regions and the outlier regions in the feature space. We evaluate this idea and show our method can train a model that outperforms existing methods with only 1% of the labeled data. We show that data synthesis also improves the model's sustainability over time and speeds up the retraining. Finally, we compare data synthesis and adversarial retraining and show they can work complementary with each other to improve the model generalizability.

Detecting Bots using Stream-based System with Data Synthesis

Tianrui Hu

(GENERAL AUDIENCE ABSTRACT)

An internet bot is a computer-controlled software performing simple and automated tasks over the internet. Although some bots are legitimate, many bots are operated to perform malicious behaviors causing severe security and privacy issues. To address this problem, machine learning (ML) models that have shown great success in building security applications are widely used in detecting bots since they can identify hidden patterns learning from data. However, many ML-based approaches are difficult to deploy since model training requires labeled data, which are expensive and time-consuming to obtain in practice, especially for security tasks. Meanwhile, the dynamic-changing nature of malicious bots means bot detection models need the continuous supply of representative labeled data to keep the models up-to-date, which makes bot detection more challenging. In this thesis, we build an ML-based bot detection system to detect advanced malicious bots in real-time by processing network traffic data. We explore using a data synthesis method to detect bots with limited training data to address the limited and unrepresentative labeled data problem. Our proposed data synthesis method synthesizes unseen (or future) bot behavior distributions to enable our system to detect bots with extremely limited labeled data. We evaluate our approach using real-world datasets we collected and show that our model outperforms existing methods using only 1% of the labeled data. We show that data synthesis also improves the model's sustainability over time and helps to keep it up-to-date easier. Finally, we show that our method can work complementary with adversarial retraining to improve the model generalizability.

Acknowledgments

I would like to thank Dr. Gang Wang and Dr. Bimal Viswanath for their unwavering guidance and valuable advice during the development of this work and mentoring me throughout my master's study. I also want to thank my committee member Dr. Bert Huang for his support and constructive comments. This project is a joint work with Steve Jan, Qingying Hao, and Jiameng Pu. Many thanks for their great contributions to this work. I am very thankful to all my colleagues at Virginia Tech. Last, I'm extremely grateful to my parents and friends for their endless support throughout the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Machine Learning in Security

In recent years, machine learning (ML) has been widely use in building security defenses to protect computer and networked systems [6, 18, 40]. Driven by empirical data, machine learning algorithms can identify hidden patterns that cannot be easily expressed by rules or signatures. This capability leads to various ML-based applications in security areas such as malicious website detection [59, 66], malicious phone call classification [35], network traffic analysis [6], malware classification [5, 13, 14, 57], and intrusion detection [24, 40].

A common challenge faced by ML-driven systems is that they often require "labeled data" to train a good detection model [5, 66]. The lack of labeled data limits the performance of many supervised models. In practice, it is highly expensive and time-consuming to obtain labels (*e.g.*, via manual efforts). For security applications, the challenge is further amplified by the dynamic behavior changes of attackers — to keep the detection models up-to-date, there is a constant need for labeling new data samples over time [33].

To solve this problem, existing works have developed unsupervised models that don't require labels[30, 73], but they are often limited in accuracy compared to supervised models. More details will be introduced in Chapter 2.3. A promising and yet under-explored direction is to perform *data synthesis*. The idea is to generate synthesized data to augment model training,

with limited data labels. Many data synthesis methods have been proposed in computer vision and natural language processing domain. However, their goal, which is to mimic the patterns that resemble training data, is different from data synthesis in the domain of security. For many of the security applications, the challenge is the lack of complete knowledge of the problem space, especially the attackers' (future) data distribution, making it difficult to properly guide the data synthesis process. The benefits and limitations of this approach remain unclear in security applications. In this thesis, we use bot detection as an example to explore the use of data synthesis to enable bot detection with limited training data.

1.2 Bot Detection

Bots are computer-controlled software that pretends to be real users to interact with online services and other users in online communities, performing simple and repetitive tasks such as web crawling. While there are bots designed for good causes (search engine crawlers, research bots) [19, 36, 52], most bots are operated to engage malicious actions such as spam, scam, click fraud and data scrapping [12, 17, 18, 20, 26, 64, 65, 68]. Increasingly malicious bots can harm the experience of legitimate users by conducting these malicious activities. While many existing efforts are devoted to bot detection, the problem is still challenging due to the dynamic changing nature of bots.

There are three main existing bot detection methods:

CAPTCHA: Online Turing Tests. CAPTCHA is short for "Completely Automated Public Turning Test to tell Computers and Humans Apart" [67]. CAPTCHA is useful to detect bots since it is designed to give a task that extremely difficult to perform by computer softwares but easy to perform by humans. Many different categories of CAPTCHA systems

have been developed and widely used over the internet. However, CAPTCHA is always limited by its coverage in practice. The reason is that aggressively delivering CAPTCHA to legitimate users would significantly hurt user experience. Although the given tasks are all simple for humans, it is annoying if users are asked to perform a task very often. Therefore, services want to deliver a minimum number of CAPTCHAs to benign users while maximizing the number to detected bots. As such, it is often used as a validation method, rather than a detection method, to verify if a suspicious user caught by other detection methods is truly a bot.

Rule-based Approaches. Rule-based detection approaches detect bots following predefined rules [56]. Rules are often hand-crafted based on defenders' domain knowledge. In practice, rules are usually designed to be highly conservative to avoid false detection on benign users. Compared with machine learning models, rules do not need training, and can provide a precise reason for the detection decision. On the other hand, they are relatively static and easy to be evaded.

Machine Learning based Approaches. Machine learning has been widely used in the security domain including bot detection. Some techniques have been proposed to improve the detection performance [16, 34, 49]. A common way is supervised training with labeled bot data and benign user data [20, 23, 32, 64]. However, they are limited by the continuous supply of representative labeled data which is expensive and time-consuming to collect as mentioned in Chapter 1.1. Additionally, it is impractical to assume that all possible malicious patterns can be captured from training data. As a result, many methods cannot adapt to changes in attack patterns. To tackle the drawbacks of supervised methods, prior work has proposed unsupervised anomaly detection methods [2, 28, 37], but they are limited by their performance. Moreover, these ML-based methods often require application-specific features and heavy feature engineering approaches, which make methods less generic, difficult to be

deployed to various web services.

To summarize, there are various challenges to deploy existing bot detection methods in practice:

- **Challenge-1: Bots are Evolving.** Bot behaviors are dynamically changing, which creates a challenge for the static rule-based system. Once a rule is set, bots might make small changes to bypass the pre-defined threshold.
- **Challenge-2: Limited Labeled Data.** Data labeling is a common challenge for supervised machine learning methods, especially when labeling requires manual efforts and when there is a constant need for new labels over time. For bot detection, CAPTCHA is a useful way to obtain "labels". However, CAPTCHA cannot be delivered to all requests to avoid degrading user experience. As such, it is reasonable to assume the training data is limited or biased.
- **Challenge-3: Generalizability.** Most bot detection methods are heavily engineered for their specific applications (*e.g.* online social networks, gaming, e-commerce websites) [20, 23, 64, 68]. Due to the use of application-specific features (*e.g.*, social graphs, user profile data, item reviews and ratings), the proposed model is hardly generalizable, and it is difficult for industry practitioners to deploy an academic system directly. Application-dependent nature also makes it difficult to share pre-trained models among services.

1.3 Overview of This Work

In this thesis, we build a ML-based bot detection system with a data synthesis method against these challenges. We obtained a real-world network traffic dataset that contains 23,000,000

network requests to three different online services (*e.g.*, e-commerce) over 3 different months in August 2018, January 2019, and September 2019 by collaborating with a security company. The "ground-truth" labels are provided by the security company's CAPTCHA system and manual verification. This dataset allows us to explore the design of a *generic* stream-based machine learning model for real-time bot detection. Our machine learning model is not redundant to the existing rule-based system and the CAPTCHA system, since the true value of a machine learning model is to handle attacker behaviors that cannot be precisely expressed by "rules". Therefore, we exclude bots that were already precisely flagged by existing rules and focus on the remaining "advanced bots" that bypassed the rules. Our system allows the defender to avoid massively delivering CAPTCHAs to real users, by guiding the CAPTCHA delivery to the likely-malicious users detected by our system. The model is designed to be generic, which only relies on basic network-level information without taking any application-level information. We design a novel feature encoding schema that allows the system to encode new traffic data as they arrive to perform real-time bot detection and work on anonymized data, further improving its portability across web services. We empirically validated that (i) well-trained machine learning models can help to detect advanced bots which significantly boosts the overall "recall" (by 15% to 30%) with a minor impact on the precision; (ii) limited training data can indeed cripple the supervised learning model, especially when facing more complex bot behaviors.

To address the problem of limited data, we explore the design of a new data synthesis method. We propose ODDS, which is short for "Outlier Distribution aware Data Synthesis". The key idea is to perform a distribution-aware synthesis based on known benign user data and *limited* bot samples. The assumption is that the benign samples are relatively more stable and representative than the limited bot data. We thus synthesize new bot samples for the unoccupied regions in the feature space by differentiating "clustered regions" and "outlier

regions". At the clustered regions (which represent common user/bot behavior), our data synthesis is designed to be conservative by gradually reducing the synthesis aggressiveness as we approach the benign region. In the outlier areas (which represent rare user behavior or new bot variants), our data synthesis is more aggressive to fill in the feature space. Based on these intuitions, we designed a customized Generative Adversarial Network (GAN) with two complementary generators to synthesize clustered and outlier data simultaneously.

We evaluate the ODDS using real-world datasets and show that it outperforms many existing methods. Using 1% of the labeled data, our data synthesis method can improve the detection performance close to that of existing methods trained with 100% of the data. Besides, we show that ODDS not only outperforms other supervised methods but improves the life-cycle of a classifier (*i.e.*, staying effective over a longer period). It is fairly easy to retrain an ODDS (with 1% of the data) to keep the models up-to-date. Furthermore, we compare data synthesis with adversarial retraining. We show that, as a side effect, data synthesis helps to improve the model resilience to blackbox adversarial examples, and it can work jointly with adversarial retraining to improve the generalizability of the trained model. Finally, we analyze the errors of ODDS to understand the limits of data synthesis.

We have three main contributions:

- *First:* we build a stream-based bot detection system to complement existing rules to catch advanced bots. The key novelty is the stream-based feature encoding scheme which encodes new data as they arrive. This allows us to perform real-time analysis and run bot detection on *anonymized* network data.

- *Second:* we describe a novel data synthesis method to enable effective model training with limited labeled data. The method is customized to synthesize the clustered data and the outlier data differently.

- *Third:* we validate our systems using real-world datasets collected from three different online services. We demonstrate the promising benefits of data synthesis and discuss the limits of the proposed method.

Chapter 2

Background

In this chapter, we introduce background knowledge related to our study.

2.1 Long Short-Term Memory Network

As a specialized Recurrent Neural Network (RNN) [31], long short-term memory network (LSTM) [27] is designed to capture the relationships of events in a sequence and has been widely applied to model sequential data [38]. The connection of nodes in an RNN allows it to process sequences of data, exploiting temporal behaviors, other than a single data point. As an updated version of RNN, LSTM is able to remember the values from earlier stages better for future use. Given a sequence of data $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$, where $\mathbf{x}_t \in \mathbb{R}^d$ denotes the input at t step, LSTM is able to exploit temporal behaviors in sequences and make predictions. LSTM has a hidden state vector $\mathbf{h}_t \in \mathbb{R}^h$ to keep track of the sequence information from the current input $\mathbf{x}_t$ and the previously hidden state $\mathbf{h}_{t-1}$. Hidden state $\mathbf{h_t}$ is computed by the input gate, forget gate, output gate. We simplify LSTM as the following equation:

$$\mathbf{h}_t = \text{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1}),$$

where $\mathbf{x}_t$ is the input of the current step; $\mathbf{h}_{t-1}$ is the hidden vector of the last step; $\mathbf{h}_t$ indicates the output of the current step. LSTM has been used in many application domains

such as natural language processing, text classification [75] and fraud detection [73]. In this thesis, we use LSTM to process sequential network traffic data in our system.

2.2 Generative Adversarial Networks

Generative adversarial networks (GAN) is first proposed by Ian Goodfellow et al [25] as a new generative model. With its strong capability of generating new data that resemble the training set, it has become one of the most popular generative models in the deep learning field. The model mainly consists of two parts, generator G and discriminator D. The high level idea of the generator is to make generated samples to match the real data distribution p_{data}. On the other hand, the discriminator D is a classifier that predicts whether an input is a real data $\mathbf{x}$ or a generated fake data from $G(z)$. Both models G and D are trained simultaneously. G learns to generate data that are difficult to classify by D, while D adapts to discriminate the generated data from the real data.

In other word, GAN is theoretically formalized as a minimax game with the value function $V(G, D)$:

$$\min_G \max_D V(G, D) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))]$$

where p_z represents the distribution of the input noise variables z, and p_{data} represents the distribution of real data.

GAN has been widely used in many application domains such as in computer vision [29, 76] and security domains such as fraud detection [73]. In this thesis, we propose a customized GAN to synthesize data to enable effective model training with limited labeled data.

2.3 Anomaly Detection

Anomaly detection aims to detect anomalous data samples compared to known data distribution [3, 51, 63, 74, 78]. In recent years, researchers have applied ML-based anomaly detection methods to detect bots and other fraudulent activities over the internet by network traffic [30, 73]. Anomaly detection approaches include unsupervised anomaly detection techniques that don't require labeled data but very sensitive to noise, and supervised anomaly detection methods that require a substantial number of labeled data (*e.g.* "normal" and "abnormal"). Many anomaly detection methods depend on an assumption that the normal/benign data should be (relatively) representative and stable so that normal regions in feature space can be distinguished from anomalous regions. This assumption is similar to the assumption of our data synthesis method that discusses in chapter 5. In this thesis, we use an anomaly detection method OCAN [73] as our baselines, showing the limitation of anomaly detection methods and the benefit of our designs of synthesizing new data based on both the normal samples and the limited abnormal samples.

2.4 Data Augmentation using GANs

To generate *more* data for training, various transformations can be applied to existing training data. In the domain of computer vision and natural language processing, researchers have proposed various data augmentation methods including GAN to improve the performance of one-shot learning [4], image segmentation [8], image rendering [55], and emotion classification [77]. However, most of these methods aren't designed to synthesize data for changing behaviors. Their goal is to mimic the pattern of existing data to compensate for the limited data. The goal of our proposed data augmentation method would not only compensate for

the limited data but also explore the unknown suspicious spaces by generating data all over the space except benign clusters. The most related work to ours is OCAN [73], which uses GAN to synthesize malicious samples for fraud detection. We compare our system with OCAN in our evaluation.

2.5 Adversarial Examples

Although machine learning models have been widely used and have outstanding performance in many applications, unfortunately, they are vulnerable to adversarial examples, which are specifically crafted examples that can fool a model. The existence of adversarial examples was first formally discussed by Szegedy et al[61]. An adversarial example is crafted by applying small but intentional perturbations to input data. Adversarial examples can bring significant security implications since they are able to fool a model to output an incorrect answer with high confidence, because of the poor generalizability and robustness of the model. For example, a popular method proposed by Carlini and Wagner [10] can generate adversarial examples by powerful attack algorithms with 100% success rate. It means attackers may evade ODDS by changing their behaviors to turn them into adversarial examples. We evaluate the performance of our system against adversarial attacks in the whitebox and blackbox setting in Chapter 6.7. The difference between whitebox attack and blackbox attack is that the whitebox setting assumes the attackers craft adversarial examples with the knowledge of the training data and the model parameters, but the blackbox setting assumes they don't know our model. Instead, the attackers can use a surrogate model to generate blackbox adversarial examples.

Adversarial examples can also be used to improve the robustness of a model by simply adding adversarial examples into the training set. The knowledge learns from adversarial

examples helps models to become more resilient, leaving fewer vulnerabilities for attackers. Besides, it also leads to better performance on the original testing set since it improves the model's generalizability on unseen distributions, which is similar to our data synthesis method. Therefore, adversarial re-training, which is using adversarial examples updating models, has been widely used to improve model performance and resilience to attacks. We discuss the relationship between adversarial re-training and our data synthesis method in Chapter 6.7.

Chapter 3

Dataset and Problem Definition

Through our collaboration with a security company, we obtained the network traffic data from three online services over three different months within a year. Each dataset contains the "ground-truth" labels on the traffic of bots and benign users. The dataset is suitable for our research for two main reasons. First, each online service has its own service-specific functionality and website structures. This offers a rare opportunity to study the "generalizability" of a methodology. Second, the datasets span a long period of time, which allows us to analyze the impact of bot behavior changes.

3.1 Data Collection

We collected data by collaborating with a security company that performs bot detection and prevention for different online services. They gather and analyze the network logs from their customers. We obtained permission to access the *anonymized* network logs from three websites.

Table 3.1 shows the summary of the datasets. For anonymity purposes, we use `A`, `B`, `C` to represent the 3 websites. For all three websites, we obtained their logs in August 2018 (08/01 to 08/31). Then we collected data in January 2019 (01/08 to 01/31) and September 2019 (09/01 to 09/30) for two websites (`A`, and `B`). We were unable to obtain data from website `C` for the January and September of 2019 due to its service termination with the company.

Table 3.1: Dataset summary.

Site	August 2018		January 2019		September 2019	
	#Request	Uniq.IP	#Request	Uniq.IP	#Request	Uniq.IP
A	2,812,355	225,331	1,981,913	157,687	1,676,842	151,304
B	4,022,195	273,383	2,559,923	238,678	5,579,243	1,301,310
C	4,388,929	180,555	-	-	-	-
All	11,223,479	667,537	4,541,836	393,504	7,256,085	1,447,247

The dataset contains a series of timestamped network requests to the respective website. Each request contains a URL, a source IP address, a referer, a cookie, a request timestamp (in milliseconds) and the browser version (extracted from User-Agent). To protect the privacy of each website and its users, only timestamp is shared with us in the raw format. All other fields including URL, IP, cookie, and browser version are shared as *hashed values*. This is a common practice for researchers to obtain data from industry partners. On one hand, this type of anonymization increases the challenges for bot detection. On the other hand, this encourages us to make more careful design choices to make sure the system works well on anonymized data. Without the need to access the raw data, the system has a better chance to be generalizable. In total, the dataset contains 23,021,400 network requests from 2,421,184 unique IP addresses.

3.2 Reprocessing: IP-Sequence

Our goal is to design a system that is applicable to a variety of websites. For this purpose, we cannot rely on the application-level user identifier to attribute the network requests to a "user account". This is because not all websites require user registration (hence the notion of "user account" does not exist). We also did not use "cookie" as a user identifier because we observe that bots often frequently clear their cookies in their requests. Using cookies makes it difficult to link the activities of the same bot. Instead, we group network requests based

on the source IP address.

Given an IP, a straightforward way might be labeling the IP as "bot" or "benign". However, such *binary* labels are not fine-grained enough for websites to take further actions (*e.g.*, delivering a CAPTCHA or issuing rate throttling). The reason is that it's common for an IP address to have both legitimate and bot traffic at different time periods, *e.g.*, due to the use of web proxy and NAT (Network Address Translation). As such, it is more desirable to make fine-grained decisions on the "sub-sequences" of requests from an IP address.

To generate *IP-sequences*, for each IP, we sort its requests based on timestamps and process the requests as a stream. Whenever we have accumulated T requests from this IP, we produce an IP-sequence. In this thesis, we empirically set $T = 30$. We perform bot detection on each IP-sequence.

3.3 Ground-truth Labels

We obtain the ground-truth labels from the CAPTCHA system and the internal rule-based systems used in the security company. Their security team also sampled both labels for manual examination to ensure the reliability.

CAPTCHA Labels. The company runs an advanced CAPTCHA system for all its customers. The system delivers CAPTCHAs to a subset of network requests. If the "user" fails to solve the CAPTCHA, that specific request will be marked with a flag. For security reasons, their selection process to deliver CAPTCHAs will not be made public. At a high level, requests are selected based on proprietary methods that aim to balance exploring the entire space of requests versus focusing on requests that are more likely to be suspicious (hence limiting impact on benign users). Given an IP-sequence, if one of the requests is

Table 3.2: Estimated false positives of rules on IP-sequences.

Website	Matched by Rules	Rules Matched & Received CAPTCHA	Solved CAPTCHA
A	42,487	38,294	4 (0.01%)
B	23,346	12,554	0 (0%)
C	50,394	19,718	0 (0%)

flagged, we mark the IP-sequence as "bot". The security team has sampled the flagged data to manually verify the labels are reliable.

We are aware that certain CAPTCHA systems are vulnerable to automated attacks by deep learning algorithms [1, 7, 41, 70]. However, even the most advanced attack [70] is not effective on all CAPTCHAs (*e.g.*, Google's CAPTCHA). In addition, recent works show that adversarial CAPTCHAs are effective against automated CAPTCHA-solving [54]. To the best of our knowledge, the CAPTCHA system used by the company is not among the known vulnerable ones. Indeed, the CAPTCHA system could still be bypassed by human-efforts-based CAPTCHA farms [42]. On one hand, we argue that human-based CAPTCHA solving already significantly increased the cost of bots (and reduced the attack scale). On the other hand, we acknowledge that CAPTCHA does not provide a complete "ground-truth".

Rule-Based Labels. Another source of labels is the company's internal rules. The rules are set to be conservative to achieve near perfect precision while sacrificing the recall (*e.g.*, looking for humanly-impossible click rate, and bot-like User-Agent and referer). To avoid giving attackers the advantage, we do not disclose the specific rules. Their security team has sampled the rule-labels for manual verification to ensure reliability. We also tried to validate the reliability on our side, by examining whether rule-labels are indeed highly precise (low or no false positives). We extract all the IP-sequences that contain a rule-label, and examined how many of them have *received* and *solved* a CAPTCHA. A user who can solve

the CAPTCHA is likely a false positive. The results are shown in Table 3.2. For example, for website A, the rules matched 42,487 IP-sequences. Among them, 38,294 sequences have received a CAPTCHA, and only in 4 out of 38,294 (0.01%) users solved the CAPTCHA. This confirms the extremely high precision of rules. As a trade-off, the rules missed many real bots, which are further discussed in Chapter 4.1.

3.4 Problem Definition

In summary, we label an IP-sequence as "bot" if it failed the CAPTCHA-solving or it triggered a rule (for those that did not receive a CAPTCHA). Otherwise, the IP-sequence is labeled as "benign". Our goal is to classify bots from benign traffic accurately at the IP-sequence level *with highly limited labeled data*. We are particularly interested in detecting bots that bypassed the existing rule-based systems, *i.e.*, advanced bots. Note that our system is not redundant to the CAPTCHA system, given that CAPTCHA can be only applied to a small set of user requests to avoid hurting the user experience. Our model can potentially improve the efficiency of CAPTCHA delivery by pinpointing suspicious users for verification.

Scope and Limitations. Certain types of attackers are out of scope. Attackers that hire human users to solve CAPTCHAs [42] are not covered in our ground-truth. We argue that pushing all attackers to human-based CAPTCHA-solving would be one of the desired goals since it would significantly increase the cost of attacks and reduce the attack speed (*e.g.*, for spam, fraud, or data scraping).

Chapter 4

Basic Bot Detection System

In this chapter, we present the *basic designs* of the bot detection system. More specifically, we want to build a machine learning model to detect the advanced bots that bypassed the existing rules. In the following, we first filter out the simple bots that can be captured by rules, and then describe our stream-based bot detection model. In this chapter, we use all the available training data to examine model performance. In the next chapter, we introduce a novel data synthesis method to detect bots with limited data (Chapter 5).

As an overview, the data processing pipeline has two steps.

- **Phase I:** Applying existing rules to filter out the easy-to-detect bots (pre-processing).
- **Phase II:** Using a machine learning model to detect the "advanced bots" from the remaining data.

4.1 Phase I: Filtering Simple Bots

As discussed in Chapter 3.3, the company's internal rules are tuned to be highly precise (with a near 0 false positive rate). As such, using a machine learning method to detect those simple bots is redundant. The rule-based system, however, has a low recall (*e.g.* 0.835 for website `B` and 0.729 for website `C`, as shown in Table 4.4). This requires Phase II to detect the advanced bots that bypassed the rules.

Table 4.1: Ground-truth data of IP-sequences.

		A	B	C
August 2018	Simple bot	42,487	23,346	50,394
	Advanced bot	6,117	2,677	19,113
	Benign	15,390	48,578	32,513
January 2019	Simple bot	30,178	10,434	-
	Advanced bot	4,245	2,794	-
	Benign	10,393	26,922	-
September 2019	Simple bot	8,974	18,298	-
	Advanced bot	15,820	9,979	-
	Benign	12,644	37,446	-

Table 4.1 shows the filtering results. Simple bots represent those bots matched by rules. Advanced bots are bots caught by CHAPTCHAs. We do not consider IPs that have fewer than $T = 30$ requests. The intuition is that, if a bot made less than 30 requests in a given month, it is not a threat to the service. We set $T = 30$ because the sequence length T needs to be reasonably large to obtain meaningful patterns [62]. As a potential evasion strategy, an attacker can send no more than 30 requests per IP, and uses a large number of IPs (*i.e.*, botnets). We argue that this will significantly increase the cost of the attacker. In addition, there are existing systems for detecting coordinated botnet campaigns [44, 46, 58] which are complementary to our goals. After filtering out the simple bots, the remaining advanced bots are those captured by CAPTCHAs. For all three websites, we have more simple bots than advanced bots. The remaining data are treated as "benign". The benign sets are typically larger than the advanced bot sets, but not orders of magnitude larger. This is because a large number of benign IP-sequences have been filtered out for having fewer than 30 requests. Keeping those short benign sequences in our dataset will only make the precision and recall look better, but it does not reflect the performance in practice (*i.e.*, detecting these benign

sequences is trivial).

4.2 Phase II: Machine Learning Model

With a focus on the advanced bots, we present the *basic design* of our detector. The key novelty is not necessarily the choice of deep neural network. Instead, it is the new *feature encoding* scheme that can work on anonymized data across services. In addition, we design the system to be stream-based, which can process network requests as they come, and make a decision whenever an IP-sequence is formed.

The goal of feature encoding is to convert the raw data of an IP-sequence into a vector. Given an IP-sequence (of 30 requests), each request has a URL hash, timestamp, referrer hash, cookie flag, and browser version hash. We tested and found the existing encoding methods did not meet our needs. For instance, one-hot encoding is a common way to encode categorical features (*e.g.*, URL, cookie). In our case, because there are hundreds of thousands of distinct values for specific features (*e.g.*, hashed URLs), the encoding can easily produce high-dimensional and sparse feature vectors. Another popular choice is the embedding method such as Word2Vec, which generates a low-dimensional representation to capture semantic relationships among words for natural language processing [39]. Word2Vec can be applied to process network traffic [69]: URLs that commonly appear at the same position of a sequence will be embedded to vendors with a smaller distance. Embedding methods are useful for *offline data processing*, and is not suitable for a real-time system. Word2Vec requires using a large and *relatively stable* dataset to generate a high quality embedding [50], but is not effective for embedding new or rare entities. In our case, we do not want to wait for months to collect the full training and testing datasets for offline embedding and detection.

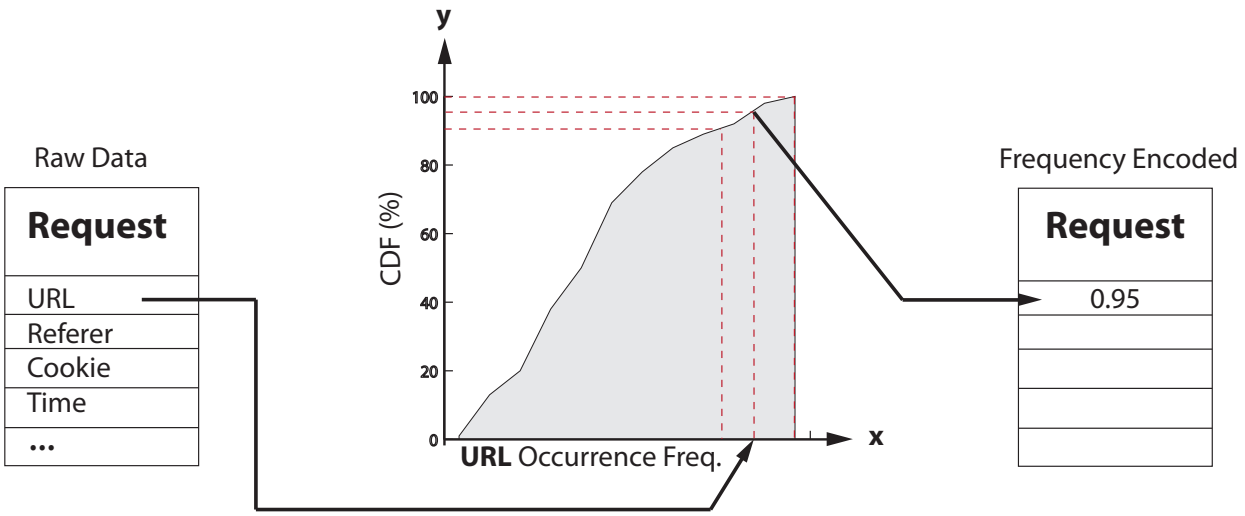


Figure 4.1: Example of frequency encoding for the visited URL.

Table 4.2: Summaries of features and their encoding scheme.

Feature	Encoding Method
URL	Frequency Distribution encoding
Referer	Frequency Distribution encoding
Browser version	Frequency Distribution encoding
Time gap	Distribution encoding
Cookie flag	Boolean

Sliding Window based Frequency Encoding. We propose an encoding method that does not require the raw entity (*e.g.*, URL) but uses the *frequency of occurrence of the entity*. The encoding is performed in a sliding window to meet the need for handling new/rare entities for real-time detection. We take "visited URL" as an example to explain how it works.

As shown in Figure 4.1, given a request, we encode the URL hash based on its occurrence frequency in the past. This example URL appears very frequently in the past at a 95-percentile. As such, we map the URL to an index value "0.95". In this way, URLs that share a similar occurrence frequency will be encoded to a similar index number. This scheme can easily handle new/rare instances: any previously-unseen entities would be assigned to a low

distribution percentile. We also don't need to manually divide the buckets but can rely on the data distribution to generate the encoding automatically. The feature value is already normalized between 0 and 1.

A key step of the encoding is to estimate the occurrence frequency distribution of an entity. For stream-based bot detection, we use a sliding window to estimate the distribution. An example is shown in Figure 4.2. Suppose IP-sequence s_1 is formed on day t (*i.e.*, the last request arrived at day t). To encode the URLs in s_1, we use the historical data in the past w days to estimate the URL occurrence distribution. Another IP-sequence s_2 is formed on day t too, and thus we use the same time window to estimate the distribution. s_3 is formed one day later on $t + 1$, and thus the time-window slides forward by one day (keeping the same window size). In this way, whenever an IP-sequence is formed, we can compute the feature encoding immediately (using the most recent data). In practice, we do not need to compute the distribution for each new request. Instead, we only need to pre-compute the distribution for each *day*, since IP-sequences on the same day share the same window.

The reasons that we apply a sliding window to estimate the distribution instead of using all data we have are first bots are evolving and second training with large a dataset is costly. The behaviors of bots a long time ago can be very different from their behaviors recently since they need to be changing to evade security measures. Therefore, the aged data, which contains outdated information, can mislead the model, making it fail to detect the latest patterns of attackers. Besides, the rapidly increasing dataset makes it more and more expensive for training if we include all historical data.

Table 4.2 shows how different features are encoded. URL, referer, and browser version are all categorical features and thus can be encoded based on their occurrence frequency. The "time gap" feature is the time gap between the current request and the previous request in the same IP-sequence. It is a numerical feature, and thus we can directly generate the

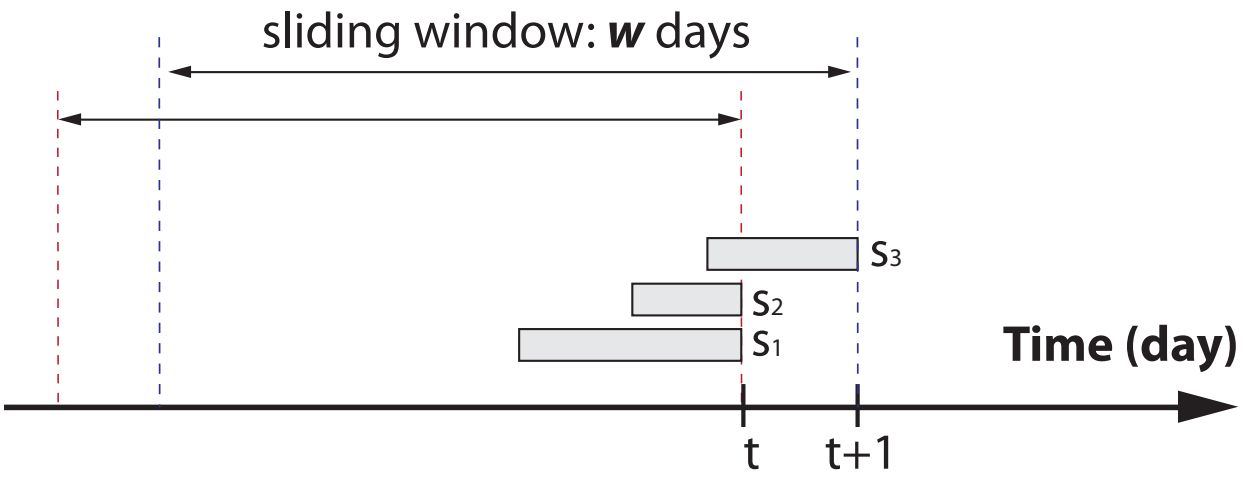


Figure 4.2: Example of sliding window for feature encoding. S_1 and S_2 are IP sequences formed on day t, and S_3 is formed on day $t+1$. The feature vendors are encoded using the past w days of data.

distribution to perform the encoding. The "cookie flag" boolean feature means whether the request has enabled a cookie. Each request has 5 features, and each IP-sequence can be represented by a matrix of 30×5 (dimension $= 150$).

Building the Classifier. Using the above features, we build a supervised Long-Short-Term-Memory (LSTM) classifier [69]. Our model contains 2 hidden LSTM layers followed by a binary classifier. The output dimension of every LSTM units in two layers is 8. Intuitively, a wider neural network is more likely to be overfitting [11], and a deeper network may have a better generalizability but requires extensive resources for training. A 2-8 LSTM model can achieve a decent balance between overfitting and training costs.

4.3 Evaluating the Performance

We evaluate our model using data from August 2018 (advanced bots). We followed the recent guideline for evaluating security-related ML models [47] to ensure result validity.

Training-Testing Temporal Split. We first ensure the *temporary training constraint* [47], which means training data should be strictly temporally precedent to the testing data. We split the August data by using the first two weeks of data for training and the later two weeks for testing. Given our feature encoding is sliding-window based, we never use the "future" data to predict the "past" events (for both bots and benign data). We did not artificially balance the ratio of bot and benign data, but kept the ratio observed from the data.

Bootstrapping the Slide Window. The sliding-window has a bootstrapping phase. For the first few days in August 2018, there is no historical data. Suppose the sliding-window size $w = 7$ days, we bootstrap the system by using the first 7 days of data to encode the IP-sequences formed in the first 7 days. On day 8, the bootstrapping is finished (sliding window is day 1 – day 7). On day 9, the window starts to slide (day 2 – day 8). The bootstrapping does not violate temporary training constraints since the bootstrapping phase is finished within the training period (the first two weeks in August).

Testing starts on day 15 (sliding window is day 7 - day 14). The window keeps sliding as we test on later days. The size of the sliding window (w) could affect the detection results. A smaller window size means the model uses less historical data to estimate the entity frequency (which could hurt the performance, especially when labeled data is sparse). However, a smaller window size also means the model uses *more recent historical data* to estimate entity frequency (which may help to improve the performance). For our experiment, we pick window size w =7 to balance the computation complexity and performance. Our dataset (a month worth of data) does not allow us to test big window sizes. The results for w =7 are shown in Table 4.3. Note that feature encoding does not require any labels. As such, we used all the data (bots and benign) to estimate the entity frequency distribution in the time window.

Table 4.3: The detection results of LSTM model on "advanced bots".

Website A			Website B			Website C		
Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
0.880	0.952	0.915	0.888	0.877	0.883	0.789	0.730	0.759

Table 4.4: The overall detection performance The "precision" and "recall" are calculated based on all bots in August 2018 (simple bots and advanced bots).

Setting	Website A		Website B		Website C	
	Precision	Recall	Precision	Recall	Precision	Recall
Rules Alone	1	0.880	1	0.835	1	0.729
Rules+LSTM	0.984	0.994	0.982	0.980	0.946	0.927

Model Performance. We compute the *precision* (the fraction of true bots among the detected bots) and *recall* (the fraction of all true bots that are detected). F1 score combines precision and recall to reflect the overall performance: $F1 = 2 \times Precision \times Recall/(Precision + Recall)$.

As shown in Table 4.3, the precision and recall of the LSTM model are reasonable, but not extremely high. For example, for website `B`, the precision is 0.888 and the recall is 0.877. The worst performance appears on `C` where the precision is 0.789 and the recall is 0.730. Since our model is focused on advanced bots that already bypassed the rules, it makes sense that they are difficult to detect.

Table 4.4 illustrates the value of the machine learning models to complement existing rules. Now we consider both simple bots and advanced bots, and examine the percentage of bots that rules and LSTM model detected. If we use rules alone (given the rules are highly conservative), we would miss a large portion of all the bots. If we apply LSTM on the remaining data (after the rules), we could recover most of these bots. The overall recall of bots can be improved significantly. For website `B`, the overall recall is booted from 0.835 to 0.980 (15% improvement). For website `C`, the recall is boosted from 0.729 to 0.927 (30%

Table 4.5: F1-score when training with limited 1% of the labeled data of the August 2018 dataset.

Website	1% Data (Avg + STD)	100% Data
A	0.904 ± 0.013	0.915
B	0.446 ± 0.305	0.883
C	0.697 ± 0.025	0.759

improvement). For website `A`, the improvement is smaller since the rules already detected most of the bots (with a recall of 0.880 using rules alone). We also show the precision is only slightly decreased. We argue that this trade-off is reasonable for web services since the CAPTCHA system can further verify the suspicious candidates and reduce false positives.

Training with Limited Data. The above performance looks promising, but it requires a large labeled training dataset. This requires aggressive CAPTCHA delivery which could hurt the benign users' experience. As such, it is highly desirable to reduce the amount of training data needed for model training.

We run a quick experiment with limited training data (Table 4.5). We randomly sample 1% of the training set in the first two weeks for model training, and then test the model on the *same testing dataset* in the last two weeks of August. Note that we sample 1% from both bots and benign classes. We repeat the experiments for 10 times and report the average F1 score. We show that limiting the training data indeed hurts the performance. For example, using 1% of the training data, `B`'s F1 score has a huge drop from 0.883 to 0.446 (with a very high standard deviation). `C` has a milder drop of 5%-6%. Only `A` maintains a high F1 score. This indicates that the advanced bots in `A` exhibit a homogeneous distribution that is highly different from benign data (later we show that such patterns do not hold over time).

On one hand, for certain websites (like `A`), our LSTM model is already effective in capturing bot patterns using a small portion of the training data. On the other hand, however, the

result shows the LSTM model is easily crippled by limited training data when the bot behavior is more complex (like `B`).

Chapter 5

Data Synthesis using ODDS

In this chapter, we explore the usage of synthesized data to augment the model training. More specifically, we only synthesize *bot* data because we expect bots are dynamically changing and bot labels are more expensive to obtain. Note that our goal is very different from the line of works on adversarial retraining (which aims to handle adversarial examples) [10, 48]. In our case, the main problem is the training data is too sparse to train an accurate model in the first place. We design a data synthesis method called ODDS. The key novelty is that our data synthesis is distribution-aware — we use different generalization functions based on the characteristics of "outliers" and "clustered data" in the labeled data samples.

5.1 Motivation of ODDS

Training with limited data tends to succumb to overfitting, leading to a poor generalizability of the trained model. Regularization techniques such as dropout and batch normalization can help, but they cannot capture the data invariance in unobserved (future) data distributions. A promising approach is to synthesize new data for training augmentation. Generative adversarial network (GAN) [25] is a popular method to synthesize data to mimic a target distribution. For our problem, however, we cannot apply a standard GAN to generate new samples that resemble the training data [71], because the input bot distribution is expected to be non-representative and changing. As such, we look into ways to *expand* the input data

distribution (with controls) to the unknown regions in the feature space.

A more critical question is, how do we know our "guesses" on the unknown distribution is correct. One can argue that it is impossible to know the right guesses without post-validations (CAPTCHA or manual verification). However, we can still leverage domain-specific heuristics to improve the chance of correct guesses. We have two assumptions. First, we assume the benign data is relatively more representative and stable than bots. As such, we can rely on benign data to generate "complementary data", *i.e.*, any data that is outside the benign region is more likely to be bots. Second, the assumption is the labeled bot data is biased: certain bot behaviors are well captured but other bot behaviors are under-represented or even missing. According to these assumptions, we need to synthesize data differently based on different internal structures of the labeled data. Our strategy is to handle high-density clustered regions and low-density outlier regions differently. In "clustered regions" in the feature space, we carefully expand the region of the already labeled bots and the expansion becomes less aggressive closer to the benign region. In the "outlier" region, we can expand the bot region more aggressively and uniformly to fill in the sparse space outside of the benign clusters.

Figure 5.1 illustrates the high level idea of the data synthesis in clustered regions and outlier regions. In the following, we design a specialized GAN with two generators for such synthesis. We name the model "Outlier Distribution aware Data Synthesis" or ODDS.

5.2 Overview of the Design of ODDS

At a high level, ODDS contains three main steps. *Step 1* is a prepossessing step to learn a latent representation of the input data. We use an LSTM-autoencoder to convert the input feature vectors into a more compressed feature space. *Step 2*: we apply DBSCAN [22] on the

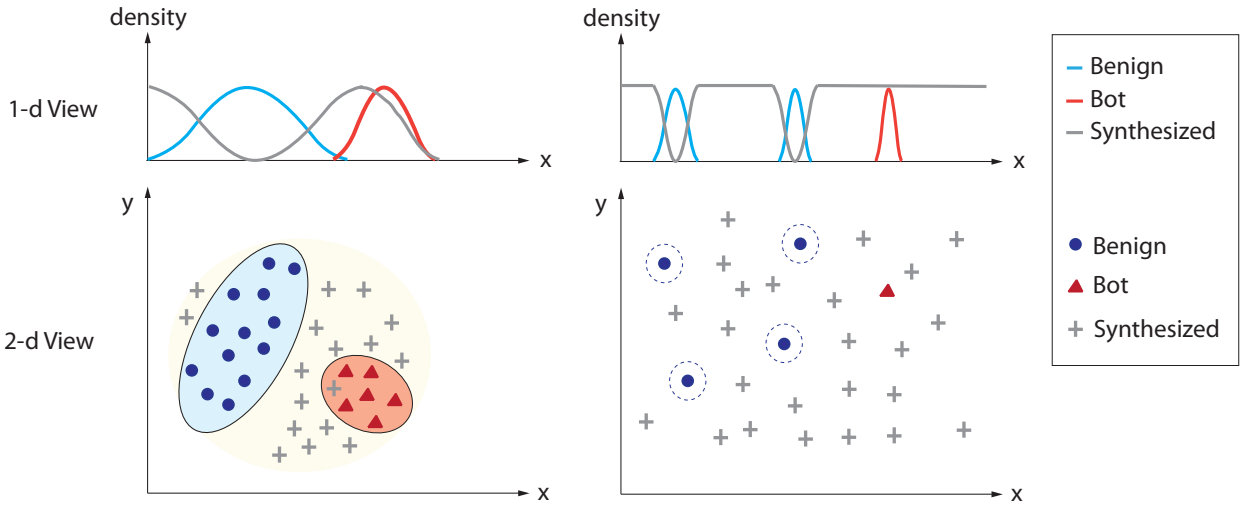


Figure 5.1: Illustrating data synthesis in the clustered data region (left) and the outlier data region (right).

new feature space to divide data into clusters and outliers. *Step 3*: we train the ODDS model where one generator aims to fit the outlier data, and the other generator fits the clustered data. A discriminator is trained to (a) classify the synthetic data from real data, and (b) classify bot data from benign data. The discriminator can be directly used for bot detection.

Step 1 and Step 2 are using well-established models, and we describe the design of Step 3 in the next section. Formally, $\mathbf{M} = \{\mathbf{X}_1, \ldots, \mathbf{X}_N\}$ is a labeled training dataset where $\mathbf{X_i}$ is an IP-sequence. $\mathbf{X_i} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ where $\mathbf{x}_t \in \mathbb{R}^d$ denotes the original feature vector of the t_{th} request in the IP-sequence.

LSTM-Autoencoder Preprocessing. This step learns a compressed representation of the input data for more efficient data clustering. LSTM-Autoencoder is a sequence-to-sequence model that contains an encoder and a decoder. The encoder computes a low-dimensional latent vector for a given input, and the decoder reconstructs the input based on the latent vector. Intuitively, if the input can be accurately reconstructed, it means the latent vector is an effective representation of the original input. We train the LSTM-Autoencoder using *all the benign data* and *the benign data only*. In this way, the autoencoder will treat bot

data as out-of-distribution anomalies, which helps to map the bot data even further away from the benign data. Formally, we convert $\mathbf{M}$ to $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N\}$, where $\mathbf{v}$ is a latent representation of the input data. We use $\mathcal{B}_v$ to represent the distribution of the latent space.

Data Clustering. In the second step, we use DBSCAN [22] to divide the data into two parts: high-density clusters and low-density outliers. DBSCAN is a density-based clustering algorithm which not only captures clusters in the data, but also produces "outliers" that could not form a cluster. DBSCAN has two parameters: s_m is the minimal number of data points to form a dense region; d_t is a distance threshold. Outliers are produced when their distance to any dense region is larger than d_t. We use the standard L_2 distance for DBSCAN. We follow a common "elbow method" (label-free) to determine the number of clusters in the data [53]. At the high-level, the idea is to look for a good silhouette score (when the intra-cluster distance is the smallest with respect to the inter-cluster distance to the nearest cluster). Once the number of clusters is determined, we can automatically set the threshold d_t to identify the outliers. DBSCAN is applied to the latent vector space $\mathbf{V}$. It is well known that the "distance function" that clustering algorithms depend on often loses its usefulness on high-dimensional data, and thus clustering in the latent space is more effective. Formally, we use DBSCAN to divide $\mathcal{B}_v$ into the clustered part $\mathcal{B}_c$ and the outlier part $\mathcal{B}_o$.

5.3 Formulation of ODDS

As shown in Figure 5.2, ODDS contains a generator G_1 for approximating the outlier distribution of $\mathcal{B}_o$, another generator G_2 for approximating the clustered data distribution $\mathcal{B}_c$, and one discriminator D.

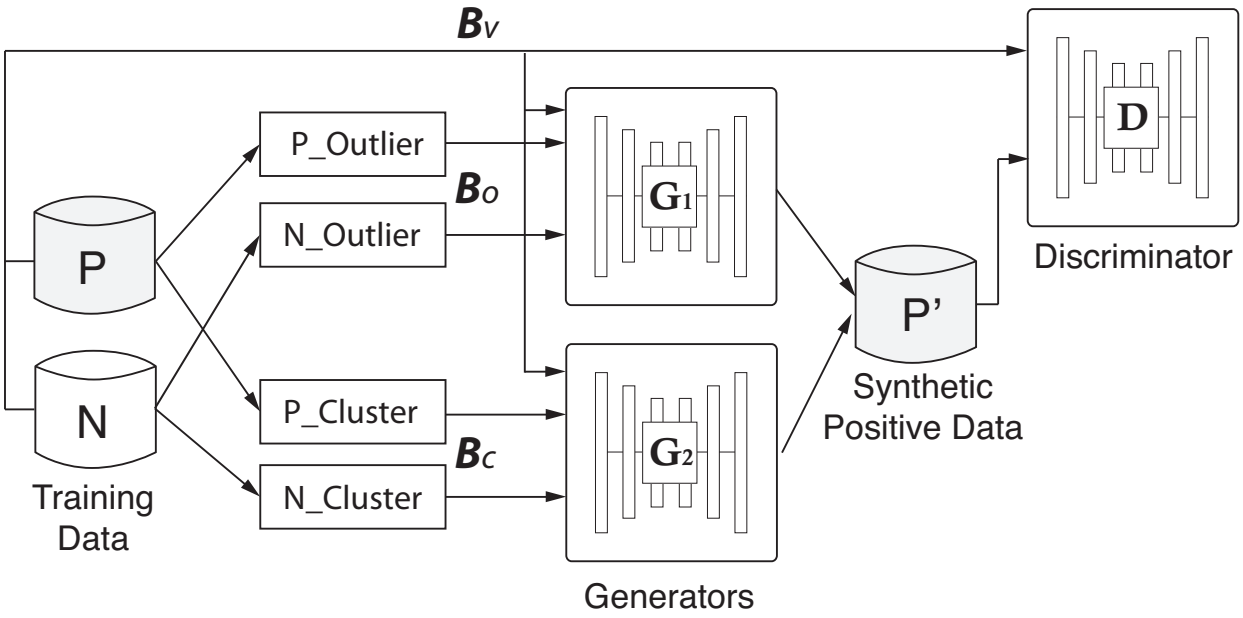


Figure 5.2: The data flow of ODDS model. "Positive (P)" represents bot data; "Negative (N)" represents benign data.

Generator 1 for Outliers. To approximate the real outliers distribution, the generator G_1 learns a generative distribution p_{G_1}. We make it close to an outlier distribution $\mathcal{O}$ that is *complementary* from the benign user representations. In other words, if the probability of the generated samples $\tilde{\mathbf{v}}$ falling in the high-density regions of benign users is bigger than a threshold $p_b(\tilde{\mathbf{v}}) > \epsilon$, it will be generated with a lower probability. Otherwise, it follows a uniform distribution to fill in the space, as shown in Figure 5.1 (right). We define this outlier distribution $\mathcal{O}$ as:

$$\mathcal{O}(\tilde{\mathbf{v}}) = \begin{cases} \frac{1}{\tau_1}\frac{1}{p_b(\tilde{\mathbf{v}})} & \text{if } p_b(\tilde{\mathbf{v}}) > \epsilon \text{ and } \tilde{\mathbf{v}} \in \mathcal{B}_v \\ C & \text{if } p_b(\tilde{\mathbf{v}}) \leq \epsilon \text{ and } \tilde{\mathbf{v}} \in \mathcal{B}_v \end{cases}$$

where ϵ is a threshold to indicate whether the generated samples are in high-density benign regions; τ_1 is a normalization term; C is a small constant; $\mathcal{B}_v$ represents the whole latent feature space (covering both outlier and clustered regions).

To learn this outlier distribution, we minimize the KL divergence between p_{G_1} and $\mathcal{O}$. Since τ_1 and C are constants, we can omit them in the objective function as follows:

$$\mathcal{L}_{KL(p_{G_1}||\mathcal{O})} = -\mathcal{H}(p_{G_1}) + \mathop{\mathbb{E}}_{\tilde{\mathbf{v}}\sim P_{G_1}} [\log p_b(\tilde{\mathbf{v}})]\, \mathbb{1}[p_b(\tilde{\mathbf{v}}) > \epsilon]$$

where $\mathcal{H}$ is the entropy and $\mathbb{1}$ is the indicator function.

We define a feature matching loss to ensure the generated samples and the real outlier samples are not too different. In other words, the generated samples are more likely to be located in the outlier region.

$$\mathcal{L}_{\text{fm}_1} = || \mathop{\mathbb{E}}_{\tilde{\mathbf{v}}\sim P_{G_1}} f(\tilde{\mathbf{v}}) - \mathop{\mathbb{E}}_{\mathbf{v}\sim \mathcal{B}_o} f(\mathbf{v})||^2$$

where f is the hidden layer of the discriminator.

Finally, the complete objective function for the first generator is defined as:

$$\min_{G_1} \quad \mathcal{L}_{KL(P_{G_1}||\mathcal{O})} + \mathcal{L}_{\text{fm}_1}$$

Generator 2 for Clustered Data. In order to approximate the real cluster distribution, the generator G_2 learns a generative distribution p_{G_2}. We make it close to a clustered bot distribution $\mathcal{C}$ where generated examples $\tilde{\mathbf{v}}$ are in high-density regions. More specifically, the clustered data is a mixture of benign and bot samples. α is the term to control whether the synthesized bot data is closer to real malicious data p_m or closer to the benign data p_b. We define this clustered bot distribution $\mathcal{C}$ as:

$$\mathcal{C}(\tilde{\mathbf{v}}) = \begin{cases} \frac{1}{\tau_2}\frac{1}{p_b(\tilde{\mathbf{v}})} & \text{if } p_b(\tilde{\mathbf{v}}) > \epsilon \text{ and } \tilde{\mathbf{v}} \in \mathcal{B}_v \\ \alpha p_b(\tilde{\mathbf{v}}) + (1-\alpha)p_m(\tilde{\mathbf{v}}) & \text{if } p_b(\tilde{\mathbf{v}}) \leq \epsilon \text{ and } \tilde{\mathbf{v}} \in \mathcal{B}_v \end{cases}$$

where τ_2 is the normalization term.

To learn this distribution, we minimize the KL divergence between p_{G_2} and $\mathcal{C}$. The objective function as follows:

$$\begin{aligned} \mathcal{L}_{KL(p_{G_2}||\mathcal{C})} = \quad & -\mathcal{H}(p_{G_2}) + \mathop{\mathbb{E}}_{\tilde{\mathbf{v}}\sim P_{G_2}}[\log p_b(\tilde{\mathbf{v}})]\mathbb{1}[p_b(\tilde{\mathbf{v}}) > \epsilon] \\ & - \mathop{\mathbb{E}}_{\tilde{\mathbf{v}}\sim P_{G_2}}[(\alpha p_b(\tilde{\mathbf{v}}) + (1-\alpha)p_m(\tilde{\mathbf{v}}))]\mathbb{1}[p_b(\tilde{\mathbf{v}}) \leq \epsilon] \end{aligned}$$

The feature matching loss $\mathcal{L}_{\text{fm}_2}$ in generator 2 is to ensure the generated samples and the real clustered samples are not too different. In other words, the generated samples are more likely to be located in the clustered region.

$$\mathcal{L}_{\text{fm}_2} = || \mathop{\mathbb{E}}_{\tilde{\mathbf{v}}\sim P_{G_2}} f(\tilde{\mathbf{v}}) - \mathop{\mathbb{E}}_{\mathbf{v}\sim \mathcal{B}_c} f(\mathbf{v})||^2$$

where f is the hidden layer of the discriminator.

The complete objective function for the second generator is defined as:

$$\min_{G_2} \quad \mathcal{L}_{KL(P_{G_2}||\mathcal{C})} + \mathcal{L}_{\text{fm}_2}$$

Discriminator. The discriminator aims to classify synthesized data from real data (a common design for GAN), and also classify benign users from bots (added for our detection purpose). The formulation of the discriminator is:

$$\min_{D} \quad \mathop{\mathbb{E}}_{\mathbf{v}\sim p_b}[\log D(\mathbf{v})] + \mathop{\mathbb{E}}_{\tilde{\mathbf{v}}\sim p_{G_1}}[\log(1 - D(\tilde{\mathbf{v}}))] + \mathop{\mathbb{E}}_{\tilde{\mathbf{v}}\sim p_{G_2}}[\log(1 - D(\tilde{\mathbf{v}}))] + \mathop{\mathbb{E}}_{\mathbf{v}\sim p_b}[D(\mathbf{v})\log D(\mathbf{v})] + \mathop{\mathbb{E}}_{\mathbf{v}\sim p_m}[\log(1 - D(\mathbf{v}))]$$

The first three terms are similar to those in a regular GAN which are used to distinguish real data from synthesized data. However, a key difference is that we do not need the discriminator to distinguish *real bot data* from *synthesized bot data*. Instead, the first three terms seek to distinguish *real benign data* from *synthesized bot data*, for bot detection. The fourth conditional entropy term encourages the discriminator to recognize real benign data with high confidence (assuming benign data is representative). The last term encourages the discriminator to correctly classify real bots from real benign data. Combining all the terms, the discriminator is trained to classify benign users from both real and synthesized bots.

Note that we use the discriminator directly as the bot detector. We have tried to feed the synthetic data to a separate classifier (*e.g.*, LSTM, Random Forest), and the results are not as accurate as the discriminator. However, there could be other classifiers or methods that can work well with our synthesized data to achieve a better performance. We leave it as a future work. In addition, using the discriminator for bot detection also eliminates the extra overhead of training a separate classifier.

Implementation. To optimize the objective function of generators, we adapt several approximations. To minimize $-\mathcal{H}(p_G)$, we adopt a pull-away term [15, 72] as an auxiliary objective. The pull-away term can help increasing the diversity of generated data and thus increase the entropy. We use it to minimize our objective function. To estimate p_m and p_b, we adopt a density estimation approach proposed by [43] which uses a supervised neural

network classifier to approximate the density.

Chapter 6

Results and Evaluation

We now evaluate the performance of ODDS. We ask the following questions: (1) how much does ODDS help when training with all the labeled data? (2) How much does ODDS help when training with limited labeled data (*e.g.*, 1%)? (3) Would ODDS help the classifier to stay effective over time? (4) Does ODDS help with classifier re-training? (5) How much contribution does each generator have to the overall performance? (6) Why does ODDS work? At what condition would ODDS offer little or no help? (7) Can ODDS further benefit from adversarial re-training?

6.1 Experiment Setup

We again focus on *advanced bots* that have bypassed the rules. To compare with previous results, we use August 2018 data as the primary dataset for extensive comparative analysis with other baseline methods. We will use the January 2019 and September 2019 data to evaluate the model performance over time and the impact on model-retraining.

Hyperparameters. Our basic LSTM model (see Chapter 4) has two LSTM hidden layers (both are of dimension of 8). The batch size is 512, the training epoch is 100, and activation function is sigmoid. Adam is used for optimization. The loss is binary crossentropy. L_2-regularization is used for both hidden layers. We use cost sensitive learning (1:2 for malicious: benign) to address the data imbalance problems.

Table 6.1: Training with 100% or 1% of the training data (the first two weeks of August-18); Testing on the last two weeks of August-18.

Method	Website A			Website B			Website C		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
RF 100%	0.896	0.933	0.914	0.831	0.594	0.695	0.795	0.669	0.727
OCAN 100%	0.891	0.935	0.912	0.659	0.882	0.732	0.878	0.543	0.671
LSTM 100%	0.880	0.952	0.915	0.888	0.877	0.883	0.789	0.730	0.759
ODDS 100%	0.897	0.940	**0.918**	0.900	0.914	**0.902**	0.832	0.808	**0.815**
RF 1%	0.877	0.836	0.856	0.883	0.202	0.343	0.667	0.636	0.651
OCAN 1%	0.855	0.951	0.901	0.680	0.736	0.707	0.650	0.344	0.450
LSTM 1%	0.866	0.946	0.904	0.601	0.355	0.446	0.694	0.701	0.697
ODDS 1%	0.859	0.943	0.900	0.729	0.845	**0.783**	0.721	0.748	**0.734**

For ODDS, the discriminator and the generators are feed-forward neural networks. All of the networks contain 2 hidden layers (100 and 50 dimensions). For generators, the dimension of noise is 50. The output of generators is of the same dimension as the output of the LSTM-autoencoder, which is 130. The threshold ϵ is set as 95_{th} percentile of the probability of real benign users predicted by a pre-trained probability estimator. We set α to a small value 0.1. We use this setting to present our main results.

Comparison Baselines. We evaluate our ODDS model with a series of baselines, including our basic LSTM model described in Chapter 4, and a non-deep learning model Random Forest [9]. We also include an anomaly detection method as a baseline. We select a GAN-based method called OCAN [73] which is recently published. The main idea of OCAN is to generate complementary malicious samples that are different from the benign data to augment the training. The key difference between OCAN and our method is that OCAN does not differentiate outliers from clustered data. In addition, as an anomaly detection method, OCAN only uses the benign data but not the malicious samples to perform the data synthesis.

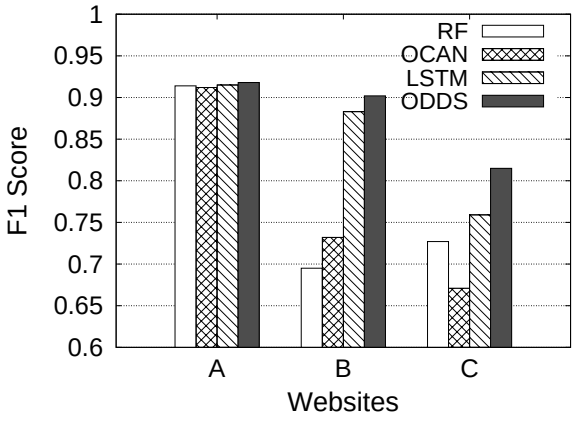


Figure 6.1: Training with 100% training data in August 2018.

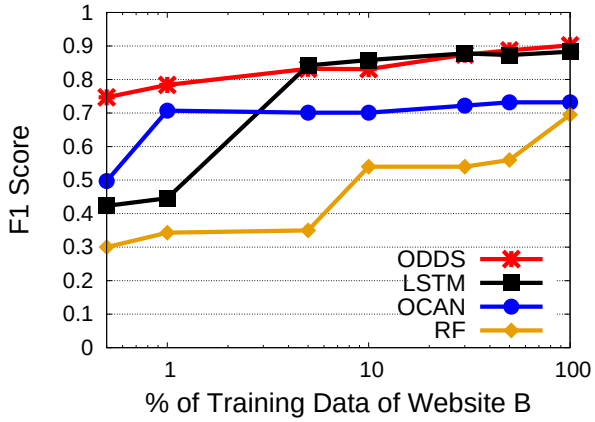
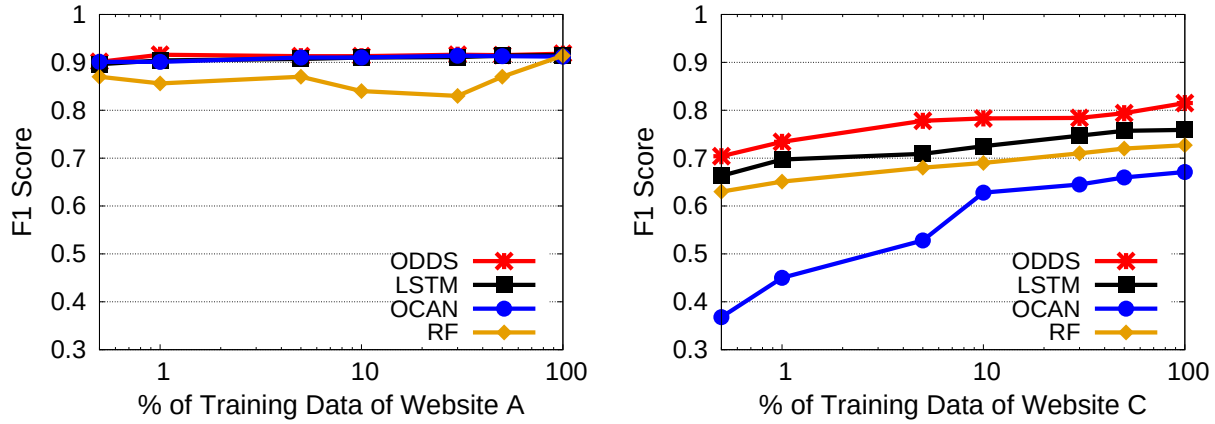


Figure 6.2: Training with x% of training data in August 2018 (B).

6.2 Training with 100% Training Data

Does ODDS help to improve the training even when training with the full labeled data?

We first run an experiment with the full-training data in August 2018 (*i.e.*, the first two weeks), and test the model on the testing data (*i.e.*, the last two weeks). Figure 6.1 shows F1 score of ODDS and other baselines. The results show that ODDS outperforms baselines in almost all cases. This indicates that, even though the full training data is relatively representative, data synthesis still improves the generalizability of the trained model on the testing data. Table 6.1 (the upper half), presents a more detailed break up of performance into precision, recall, and F1 score. The most obvious improvement is on website B where ODDS improves the F1 score by 2%-20% compared to the other supervised models. The F1 score of C is improved by 5%-14%. For website A, the improvement is minor. Our LSTM model is the second-best performing model. OCAN, as a unsupervised method, performs reasonably well compared with other supervised methods. Overall, there is a benefit for data synthesis even when there is sufficient training data.



(a) Website A (b) Website C

Figure 6.3: Training with x% of training data in August-18 (`A` and `C`).

6.3 Training with Limited Data

How much does ODDS help when training with limited training data?

As briefly shown in Chapter 4.3, the performance of the LSTM model degrades a lot when training with 1% of the data, especially for website `B`. Here, we repeat the same experiment and compare the performance of ODDS and LSTM.

Figure 6.2 shows the average F1 score on website `B`, given different sampling rates of the August training data. We can observe a clear benefit of data synthesis. The red line represents ODDS, which maintains a high level of detection performance despite the limited training samples. ODDS has an F1 score of 0.784 even when training with 1% data. This is significantly better than LSTM whose F1 score is 0.446 on 1% of training data. In addition to the average F1 score, the standard deviation of the F1 score is also significantly reduced (from 0.305 to 0.09). In addition, we show ODDS also outperforms OCAN where ODDS has a higher F1 score over all the different sampling rates. As shown in the bottom half of Table 6.1, the performance gain is mostly coming from "recall", indicating the synthesized data is helpful to detect bots in the unknown distribution.

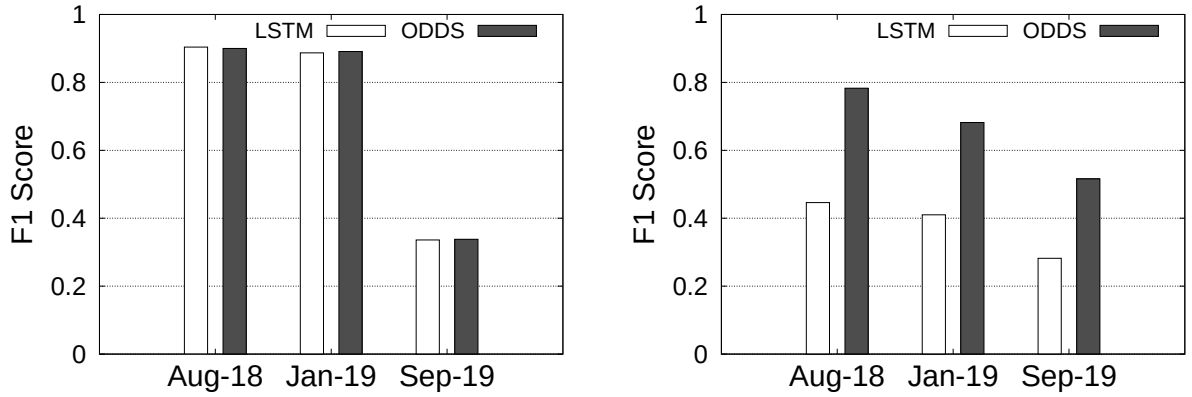
Table 6.2: Characterizing different website datasets (August 2018).

WebSite	Avg. Distance Between Benign and Bot (training)	Avg. Distance Between Train and Test (bots)
A	0.690	0.237
B	0.343	0.358
C	0.349	0.313

Figure 6.3 shows the results from other two websites where the gain of ODDS is smaller compared to that of website `B`. Website `C` still has more than 5% gain over LSTM and other baselines, but the gain is diminished in `A`. We suspect that such differences are rooted in the different bot behavior patterns in respective websites. To validate this hypothesis, we run statistical analysis on the August data for the three websites. The results are shown in Table 6.2. First, we compute the average Euclidean distance between the bot and benign data points in the August training set (averaged across all bot-benign pairs). A larger average distance indicates that bots and benign users are further apart in the feature space. The result shows that `A` clearly has a larger distance than that of `B` and `C`. This confirms that bots in `A` are already highly different from benign users, and thus it is easier to capture behavioral differences using just a small sample of the data. We also calculate the average distance between the bots in the training set and the bots in the testing set. A higher distance means that the bots in testing data have behaved more differently from those in the training data (and thus are harder to detect). We find `A` has the lowest distance, suggesting bot behaviors remain relatively consistent. `B` has the highest distance, which requires the detection model to generalize well in order to capture the new bot behaviors.

6.4 Generalizability in the Long Term

Would ODDS help the classifier stay effective for a long period of time?



(a) Website A

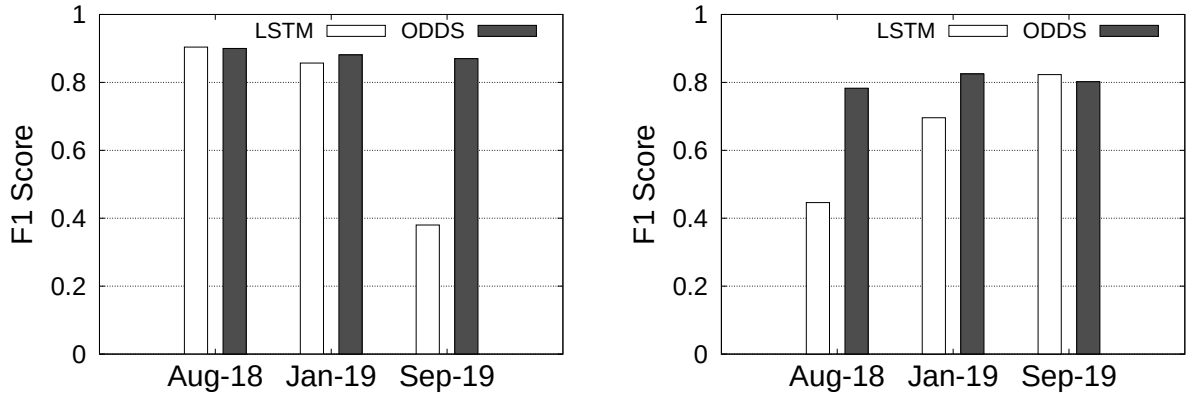
(b) Website B

Figure 6.4: The model is trained once using 1% August-18 training dataset. It is tested on August-18 testing dataset (last two weeks), and January-19 and September-19 datasets.

Next, we examine the generalizability of our model in the longer term. More specifically, we train our model using only 1% of the training dataset of August 2018 (the first two weeks), and then test the model directly on the last two weeks of August 2018, and the full months of January 2019 and September 2019. The F1 score of each testing experiment is shown in Figure 6.4. Recall that Website `C` does not have the data from January 2019 or September 2019, and thus we could only analyze `A` and `B`. As expected, the model performance decays, but in a different way between `A` and `B`. For `A`, both ODDS and LSTM are still effective in January 2019 (F1 scores are above 0.89), but become highly inaccurate in September 2019. This suggests that the bots in `A` have a drastic change of behaviors in September 2019. For website `B`, the model performance is gradually degrading over time. This level of model decay is expected given that training time and the last testing time are more than one year apart. Still, we show that ODDS remains more accurate than LSTM, confirming the benefit of data synthesis.

Does ODDS help with classifier re-training?

A common method to deal with model decay is re-training. Here, we assume the defender



(a) Website A

(b) Website B

Figure 6.5: The model is initially trained with 1% of August-18 training data, and is then re-trained each month by adding the first 1% of the training data of each month.

can retrain the model with the first 1% of the data in the respective month. We use the first 1% (instead of random 1%) to preserve the temporal consistency between training and testing (*i.e.*, never using future data to predict the past event). More specifically, the model is initially trained with only 1% of August-2018 training dataset (first two weeks) and tested in the last two weeks of August 2018. Once it comes to January 2019, we add the first 1% of the January 2019 data to the original 1% of August 2018 training data to re-train the model. This forms a January model, which is then tested on the rest of the data in January. Similarly, in September 2019, we add the first 1% of the data in September to the training dataset to retrain the model. In practice, one can choose to gradually remove/expire older training data for each retraining. We did not simulate data expiration in our experiments since we only have three months of data.

As shown in Figure 6.5 the performances bounce back after model retraining with only 1% of the data each month. In general, ODDS is better than LSTM after retraining. For example, for September 2019 of website `A`, ODDS's F1 score increases from 0.546 to 0.880 after retraining. In comparison, LSTM's F1 score is only 0.377 after the retraining. This

Table 6.3: F1 score when using only one generator; training with 100% of the training dataset of August 2018.

Website	G1 (outlier)	G2 (clusters)	Both generators
A	0.915	0.915	0.918
B	0.852	0.860	0.902
C	0.721	0.739	0.815

Table 6.4: Case study for website `B`; number of false positives and false negatives from the cluster and outlier regions; Models are trained with 1% of the training dataset of August 2018.

Cluster	Test Dataset		LSTM		ODDS	
	Malicious	Benign	FN	FP	FN	FP
Outliers	1,492	1,384	703	599	196	611
Clusters	494	26,920	287	0	102	0

suggests that data synthesis is also helpful to retrain the model with limited data.

6.5 Contribution of Generators

How much contribution does each generator have to the overall performance boost?

A key novelty of ODDS is to include two generators to handle outlier data and clustered data differently. To understand where the performance gain is coming from, we set ODDS to use only one of the generators and repeat the experiments using the August 2018 dataset (trained with 100% of the training dataset). As shown in Table 6.3, using both generators is always better than using either G1 (synthesizing outlier data) or G2 (synthesizing clustered data) alone. The difference is more obvious on website `B` since its bot data is much more difficult to separate from the benign data.

Table 6.5: Statistics about false positives (FP) and false negatives (FN) of ODDS. We calculate their average distance to the malicious and benign regions in the entire dataset, and the % of benign data points among their 100 nearest neighbors.

	Avg distance to benign	Avg distance to malicious	% benign points among 100 Nearest Neighbors
FN	0.251	0.329	100%
FP	0.644	0.402	82.0%

6.6 Insights into ODDS: Why it Works and When it Fails?

At what condition does ODDS offer little or no help?

Data synthesis has its limitations. To answer this question, we analyze the errors produced by ODDS, including false positives (FP) and false negatives (FN). In Table 6.4, we focus on the 1%-training setting for August 2018. We examine the errors located in the outlier and the clustered regions. More specifically, we run DBSCAN on the entire August 2018 dataset to identify the clustered and outlier regions. Then we retrospectively examine the number of FPs and FNs made by LSTM and ODDS (training with 1% data). We observe that ODDS's performance gain is made mainly by reducing the FNs, *i.e.*, capturing bots that LSTM fails to catch in both clustered and outlier regions. For example, FNs are reduced from 703 to 196 in outliers. The corresponding sacrifice on false positives is small (FP rate is only increased from 2.0% to 2.2%). Note that all the FPs are located in the outlier region.

To understand the characteristics of these FPs and FNs, we present a statistical analysis in Table 6.5. For all the FNs produced by ODDS, we calculate their average distance to all the benign points and malicious points in the August-18 dataset. We find that FNs are closer to the benign region (distance 0.251) than to the malicious region (distance 0.329). This indicates that bots missed by ODDS behave more similarly to benign users. We further identify

100 nearest neighbors for each FN. Interestingly, for all FNs, 100 out of their 100 nearest neighbors are benign points. This suggests that these FN-bots are completely surrounded by benign data points in the feature space, which makes them very difficult to detect.

In Table 6.5, we also analyzed the FPs of ODDS. We find that FPs are closer to the malicious region (0.402) than to the benign region (0.644), which explains why they are misclassified as "bots". Note that both 0.644 and 0.402 are high distance values, confirming that FPs are outliers far away from other data points. When we check their 100 nearest neighbors, we surprisingly find that 82% of their nearest neighbors are benign. However, a closer examination shows that most of these benign neighbors turn out to be *other FPs*. If we exclude other FPs, only 9% of their 100 nearest neighbors are benign. This confirms that FPs misclassified by ODDS behave differently from the rest of the benign users.

In summary, we demonstrate the limitation of ODDS in capturing (1) bots that are deeply embedded in the benign region, and (2) outlier benign users who behave very differently from the majority of the benign users. We argue that bots that perfectly mimic benign users are beyond the capacity of any machine learning method. It is possible that attackers could identify such "behavior regions", but there is a cost for attackers to implement such behaviors (*e.g.*, bots have to send requests slowly to mimic benign users). Regarding the "abnormal" benign users that are misclassified, we can handle them via CAPTCHAs. After several successfully-solved CAPTCHAs, we can add them back to the training data to expand ODDS's knowledge about benign users.

6.7 Adversarial Examples and Adversarial Retraining

Can ODDS benefit from adversarial re-training?

Table 6.6: Applying adversarial training on LSTM and ODDS. We use August-18 dataset from website B; Models are trained with 1% of the training dataset.

	Adversarial Retraining?	F1 score on original test set	Testing accuracy on adversarial examples
LSTM	No	0.446	0.148
ODDS	No	0.783	0.970
LSTM	Yes	0.720	1.000
ODDS	Yes	0.827	1.000

While our goal is different from adversarial re-training, we want to explore if ODDS can be further improved by adversarial re-training. More specifically, we use a popular method proposed by Carlini and Wagner [10] to generate adversarial examples, and then use them to retrain LSTM and ODDS. We examine if the re-trained model performs better on the original testing sets and the adversarial examples.

We use the August-18 dataset from B, and sample 1% of the training data to train LSTM and ODDS. To generate adversarial examples, we simulate a *blackbox attack*: we use the same 1% training data to train a CNN model which acts as a surrogate model to generate adversarial examples (Carlini and Wagner's attack is designed for CNN). Given the transferability of adversarial examples [45], we expect the attack should work on other deep neural networks trained on this dataset. We use the L2 attack to generate adversarial examples only for the *bot data* to simulate evasion. The adversarial perturbations are applied to the input feature space, i.e., after feature engineering. We generate 600 adversarial examples based on the same 1% bot training samples with different noise levels (number of iterations is 500–1000, learning rate is 0.005, confidence is set to 0–0.2). We use half of the adversarial samples (300) for adversarial retraining, *i.e.*, adding adversarial examples back to the training data to retrain LSTM and ODDS. We use the remaining adversarial examples for testing (300).

Table 6.6 shows the results. Without adversarial re-training, LSTM is vulnerable to the adversarial attack. The testing accuracy on adversarial examples is only 0.148, which means

85.2% of the adversarial examples are misclassified as benign. Interestingly, we find that ODDS is already quite resilient to the blackbox adversarial examples with a testing accuracy of 0.970. After applying adversarial-retraining, both LSTM and ODDS perform better on the adversarial examples, which is expected. In addition, adversarial-retraining also leads to better performance on the *original testing set* (the last two weeks of August-18) for both LSTM and ODDS. Even so, LSTM with adversarial-retraining (0.720) is still not as good as ODDS without adversarial retraining (0.783). The result suggests that adversarial retraining and ODDS both help to improve the model's generalizability on unseen bot distributions, but in different ways. There is a benefit to apply both to improve the model training.

Note that the above results do not necessarily mean ODDS is completely immune to all adversarial attacks. As a quick test, we run a *whitebox* attack assuming the attackers know both the training data and the model parameters. By adapting the Carlini and Wagner attack [10] for LSTM and ODDS's discriminator, we directly generate 600 adversarial examples to attack the respective model. For our discriminator, adversarial perturbations are applied in the latent space, i.e., on the output of the autoencoder. Not too surprisingly, whitebox attack is more effective. For LSTM, the testing accuracy of adversarial examples drops from 0.148 to 0. For ODDS's discriminator, the testing accuracy of adversarial examples drops from 0.970 to 0.398.

To realize the attack in practice, however, there are other challenges. For example, the attacker will need to determine the perturbations on the real-world network traces, and not just in the feature space. This is a challenging task because the data is sequential (discrete inputs) where each data point is multi-dimensional (*e.g.*, covering various metadata associated with a HTTP request). In addition, bot detection solution providers usually keep their model details confidential, and deploy their models in the cloud without exposing a public API for direct queries. These are non-trivial problems and we leave further explorations to

future work.

Chapter 7

Discussion

In this chapter, we first discuss the implications for the rule-based system and the CAPTCHA system. Then, we point out a few limitations exist in our study. Last, we briefly introduce future works and open questions.

7.1 Implications

Rules vs. Machine Learning Model. We argue that rule-based system should be the first choice over machine learning for bot detection. Compared with machine learning models, rules do not need training, and can provide a precise reason for the detection decision (*i.e.*, interpretable). Machine learning model is useful to capture more complex behaviors that cannot be accurately expressed by rules. In this work, we apply machine learning (ODDS) to detect bots that have bypassed the rules. In this way, the rules can afford to be extremely conservative (*i.e.*, highly precise but has a low recall).

CAPTCHA System. ODDS could also allow the CAPTCHA system to be less aggressive, especially on benign users. We still recommend delivering CAPTCHAs to bots flagged by rules or ODDS since there is no cost (on users' expense) for delivering CAPTCHAs to true bots. The only cost is the small number of false positives produced by ODDS, *i.e.*, benign users who need to solve a CAPTCHA. As shown in Table 6.1, the false positive is small (*e.g.*,

1-2% of benign users' requests). By guiding the CAPTCHA delivery to the likely-malicious users, ODDS allows the defender to avoid massively delivering CAPTCHAs to real users.

7.2 Limitations

Our study has a few limitations. (1) While ODDS is designed to be generic, we haven't tested it on other web services or beyond bot detection applications. Our method also relies on the assumption that benign data is relatively stable and representative, which may not hold for some cases. (2) While our "ground-truth" already represents a best-effort, it is still possible to have a small number of wrong labels. For example, in the benign set, there could be true bots that use crowdsourcing services to solve CAPTCHAs or bots that never received CAPTCHAs before. (3) Due to limited data (three disconnected months), we could not fully evaluate the robustness and effectiveness of our system over a continuous time space. The studies of sliding window and model retraining are also limited by the dataset. There are still some questions regarding the deployment of our system in practice that we didn't discuss in this thesis. (4) We make detection decisions on IP-sequences. However, it is possible that multiple users may use the same IP behind NAT/proxy. If a user chooses to use a proxy that is heavily used by attackers, we argue that it's reasonable for the user to receive some CAPTCHAs as a consequence. (5) Although ODDS improves model training with limited data, it does not mean attackers cannot evade ODDS by changing their behaviors. Attackers could identify adversarial examples in the whitebox setting or perform other adversarial attacks. (6) ODDS needs to be retrained from scratch when new bot examples become available, which has a cost.

7.3 Future Works and Open Questions

We propose a new machine learning based bot detection system and provide a preliminary understanding of using data synthesis to address the limited labeled data problem for bot detection. There also exist some future works and open questions beyond our study.

(1) Future works can test our system on other online services and other applications to evaluate the generalizability of our system. Studies on other applications would provide a more comprehensive understanding of the usage of data synthesis in the field of security. It also leads us to explore how to improve the design of ODDS, which based on two assumptions. However, our assumptions may not hold in some scenario, which requires a new designs of ODDS, such as when the benign set is also highly dynamic (*e.g.* website updates may cause benign users changing behaviors).

(2) There are some drawbacks using IP as the identifier to group requests into sequences as we state in the previous section. We can't explore more options with our dataset. Future works can further explore whether using other user identifiers to attribute the requests works better though it may hurt the generalizability of the system.

(3) A longitudinal study over a continuous long time can better evaluate the robustness and effectiveness of our system. It can provide more insights into the behaviors changing of attackers that our disconnected dataset cannot provide. Measurements about the relationship between behaviors changing, model decay, and concept drift over a long period are worth further study, which may lead to a new direction of defense system designs. It would also allow us to evaluate the impact of sliding window and model retraining. The window size is chosen as 7 since our current dataset (a month worth of data) does not support very large window sizes. There may exist a bigger window size that can achieve even better results, which is an under-explored question in our study. With a dataset over a continuous long

period, future works may look into how often services should perform model retraining and how much data is needed each time. There are still many open questions about how to handle behaviors changing and concept drifts while retraining. Especially when websites change, it could result in significant behaviors changing of benign users, which violates our assumption. Future research may explore how to better deal with behavior changes of both malicious and benign users over time.

(4) To reduce the cost of model retraining, a future direction of improvement is to perform incremental online learning [21] for model updates. Besides, incremental online learning can also help with efficiently correcting our model after identifying FPs and FNs by the CAPTCHA system.

(5) As we discussed in Chapter 6.6, we identify some FPs who behave very differently from the majority of the benign users and FNs that deeply embedded in the benign region. Future works may do more cases study about these users measuring what kind of malicious users can evade our defense and what's the cost of it. Knowing what kind of "abnormal" benign users are misclassified could help us to improve our system design to reduce the FP rate, especially in outlier regions. The step 1 and 2 of our ODDS (i.e, converting input into latent representation and dividing data into clusters and outliers) may also cause inevitable FPs and FNs due to the imperfect representations learned by LSTM-autoencoder and clustered feature spaces identified by DBSCAN. It is possible to reduce the FP and FN rate if we use other representation learning and data clustering methods to generate more effective representations and divide the data into better-quality clusters and outliers regions, which needs further study.

(6) We also believe that our methods could be integrated with or work complementary to other methods. A promising future research would be trying to feed our synthesized data into other state-of-the-art methods. For example, some semi-supervised methods are showing

great success in solving the limited data problem. Our generated data may further improve their performance.

(7) Adversarial machine learning is an active research field. Our system, like other machine learning based systems, is subject to potential evasion attacks and poisoning attacks. Although we test simple blackbox and whitebox attacks on our system and discuss the relationship between our synthesis method and adversarial retraining in Chapter 6.7, there are still some open questions. Future research may look into how ODDS can better help to deal with all kinds of adversarial attacks. Also, bot detection is not a typical adversarial machine learning problem because the "small changes" defined by the distance function in the feature space do not necessarily reflect the real-world costs to attackers [60]. For example, a simple way of evasion might be editing the "time-gap" feature, but it requires the attacker to dramatically slow down their request sending rate. Therefore, "cost-aware" adversarial evasion that realizes the attack instances in the real world becomes an interesting direction of future work. For example, a challenging question is how to translate the feature vectors back to real network traffic traces, given that our feature vectors are multi-dimensional sequential data. Moreover, how to perform an attack, either blackbox or whitebox attack is challenging for attackers since a detection model should not be exposed by the online services. It is still an open question that how to effectively attack the unexposed model in practice. Recently, researchers are also trying to hide a deployed machine learning model to defend the attacks, which can also benefit our system.

For poisoning attacks, it is possible to inject mislabeled data to the training set to affect the model training. However, it is costly to get the injected data to be considered as part of the training data in practice. For example, to inject malicious data with benign labels, it means attackers need to pay human labor to solve CAPTCHAs. Future works can study the impact of poisoning attacks on our system and bot detection problems.

(8) Another direction to solve the limited data problem for a machine learning based system is knowledge sharing (i.e., data and model sharing). For example, researchers always use state-of-the-art pre-trained models to improve their model's performance in the domains of computer vision and natural language processing. In our scenario, attackers' behaviors could be different among different services. It means the knowledge transfer from a source domain could improve the model's generalizability in a target domain to better identify unseen attackers. However, there are a few challenges in terms of knowledge sharing for security tasks. First, due to privacy concerns in the exposure of personal information that could be caused by raw data and model sharing, most of the services are not willing to share them with other services. Second, the raw feature spaces among different services are heterogeneous. The knowledge from one service could be meaningless to another since different services always have different URLs, user IPs, etc, with little overlap. Finding useful patterns to share is challenging. Our proposed frequency based feature encoding approach could allow us to explore data and pre-trained models sharing among services. It can generate transferable features that share similar feature spaces and reduce the risk of privacy exposure. However, it doesn't mean that our method can solve the challenges. The privacy issues and the effectiveness of knowledge sharing using our encoding scheme are still under-explored. It is not trivial to investigate whether our method can create feature mappings between related instances to reduce data heterogeneity. There are also more open questions like how to use shared data and model efficiently. Would existing transfer learning techniques help? Besides, only sharing the synthesized data generated from one domain to other domains may reduce privacy risks further, but also cause more challenges and questions. We leave them to future work.

Chapter 8

Conclusion

In this thesis, we propose a stream-based bot detection system to perform real-time analysis and augment it with a novel data synthesis method called ODDS. We evaluate our system on three different real-world online services, showing that it is able to capture more complex behaviors that cannot be accurately expressed by rules. We validate that ODDS makes it possible to train a good model with only 1% of the labeled data, and helps the model to sustain over a long period of time with low-cost retraining. We also explore the relationship between data synthesis and adversarial re-training, and demonstrate the different benefits from both approaches.

Bibliography

[1] William Aiken and Hyoungshick Kim. POSTER: DeepCRACk: Using deep learning to automatically crack audio CAPTCHAs. In *Proc. of ASIACCS*, 2018.

[2] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: A survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.

[3] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proc. of KDD Workshop*, 2013.

[4] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *CoRR abs/1711.04340*, 2017.

[5] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. DREBIN: Effective and explainable detection of android malware in your pocket. In *Proc. of NDSS*, 2014.

[6] Karel Bartos, Michal Sofka, and Vojtech Franc. Optimized invariant representation of network traffic for detecting unseen malware variants. In *Proc. of USENIX Security*, 2016.

[7] Kevin Bock, Daven Patel, George Hughey, and Dave Levin. unCaptcha: A low-resource defeat of reCaptcha's audio challenge. In *Proc. of WOOT*, 2017.

[8] Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger Gunn, Alexander Hammers, David Alexander Dickie, Maria Valdés Hernández, Joanna Wardlaw, and

Daniel Rueckert. GAN augmentation: Augmenting training data using generative adversarial networks. *CoRR abs/1810.10863*, 2018.

[9] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[10] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *Proc. of IEEE S&P*, 2017.

[11] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proc. of DLRS*, 2016.

[12] Ken Chiang and Levi Lloyd. A case study of the rustock rootkit and spam bot. *HotBots*, 7(10-10):7, 2007.

[13] Scott E. Coull and Christopher Gardner. Activation analysis of a byte-based deep neural network for malware classification. In *Proc. of DLS workshop*, 2019.

[14] George Dahl, Jay Stokes, Li Deng, and Dong Yu. Large-scale malware classification using random projections and neural networks. In *Proc. of ICASSP*, 2013.

[15] Zihang Dai, Zhilin Yang, Fan Yang, William W. Cohen, and Ruslan Salakhutdinov. Good semi-supervised learning that requires a bad GAN. In *Proc. of NeurIPS*, 2017.

[16] Dimitrios Damopoulos, Sofia A Menesidou, Georgios Kambourakis, Maria Papadaki, Nathan Clarke, and Stefanos Gritzalis. Evaluation of anomaly-based IDS for mobile devices using machine learning classifiers. *Security and Communication Networks*, 5(1): 3–14, 2012.

[17] Vacha Dave, Saikat Guha, and Yin Zhang. Measuring and fingerprinting click-spam in Ad networks. In *Proc. of SIGCOMM*, 2012.

[18] Vacha Dave, Saikat Guha, and Yin Zhang. ViceROI: Catching click-spam in search ad networks. In *Proc. of CCS*, 2013.

[19] Clayton Allen Davis, Onur Varol, Emilio Ferrara, Alessandro Flammini, and Filippo Menczer. BotOrNot: A system to evaluate social bots. In *Proc. of WWW*, 2016.

[20] Emiliano De Cristofaro, Nicolas Kourtellis, Ilias Leontiadis, Gianluca Stringhini, and Shi Zhou. LOBO: Evaluation of generalization deficiencies in Twitter bot classifiers. In *Proc. of ACSAC*, 2018.

[21] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. Lifelong anomaly detection through unlearning. In *Proc. of CCS*, 2019.

[22] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of KDD*, 1996.

[23] David Freeman, Sakshi Jain, Markus Dürmuth, Battista Biggio, and Giorgio Giacinto. Who are you? A statistical approach to measuring user authenticity. In *Proc. of NDSS*, 2016.

[24] Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R Kulkarni, and Prateek Mittal. SAQL: A stream-based query system for real-time abnormal system behavior detection. In *Proc. of USENIX Security*, 2018.

[25] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proc. of NeurIPS*, 2014.

[26] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering

analysis of network traffic for protocol- and structure-independent botnet detection. In *Proc. of NDSS*, 2008.

[27] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[28] Luca Invernizzi, Paolo Milani Comparetti, Stefano Benvenuti, Christopher Kruegel, Marco Cova, and Giovanni Vigna. EVILSEED: A guided approach to finding malicious web pages. In *Proc. of IEEE S&P*, 2012.

[29] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proc. of CVPR*, 2017.

[30] Gregoire Jacob, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. PUBCRAWL: Protecting users and businesses from crawlers. In *Proc. of USENIX Security*, 2012.

[31] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. 1999.

[32] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. In *Proc. of BICT*, 2016.

[33] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *Proc. of USENIX Security*, 2017.

[34] Sneha Kudugunta and Emilio Ferrara. Deep neural networks for bot detection. *Information Sciences*, 467:312–322, 2018.

[35] Huichen Li, Xiaojun Xu, Chang Liu, Teng Ren, Kun Wu, Xuezhi Cao, Weinan Zhang, Yong Yu, and Dawn Song. A machine learning approach to prevent malicious calls over telephony networks. In *Proc. of IEEE S&P*, 2018.

[36] Tetyana Lokot and Nicholas Diakopoulos. News bots: Automating news and information dissemination on Twitter. *Digital Journalism*, 4:682–699, 8 2016.

[37] Emaad Manzoor, Sadegh M Milajerdi, and Leman Akoglu. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proc. of KDD*, 2016.

[38] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proc. of INTERSPEECH*, 2010.

[39] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proc. of NeurIPS*, 2013.

[40] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Proc. of NDSS*, 2018.

[41] Manar Mohamed, Niharika Sachdeva, Michael Georgescu, Song Gao, Nitesh Saxena, Chengcui Zhang, Ponnurangam Kumaraguru, Paul C. van Oorschot, and Wei-Bang Chen. A three-way investigation of a game-CAPTCHA: Automated attacks, relay attacks and usability. In *Proc. of ASIACCS*, 2014.

[42] Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. Re: CAPTCHAs: Understanding CAPTCHA-solving services in an economic context. In *Proc. of USENIX Security*, 2010.

[43] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proc. of ICML*, 2005.

[44] Shirin Nilizadeh, Hojjat Aghakhani, Eric Gustafson, Christopher Kruegel, and Giovanni Vigna. Think outside the dataset: Finding fraudulent reviews using cross-dataset analysis. In *Proc. of WWW*, 2019.

[45] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. Transferability in machine learning: From phenomena to black-box attacks using adversarial samples. *CoRR abs/1605.07277*, 2016.

[46] Abhinav Pathak, Feng Qian, Y. Hu, Zhuoqing Mao, and Supranamaya Ranjan. Botnet spam campaigns can be long lasting: Evidence, implications, and analysis. In *Proc. of SIGMETRICS*, 2009.

[47] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *Proc. of USENIX Security*, 2019.

[48] Yao Qin, Nicholas Carlini, Ian Goodfellow, Garrison Cottrell, and Colin Raffel. Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. In *Proc. of ICML*, 2019.

[49] Supranamaya Ranjan, Joshua Robinson, and Feilong Chen. Machine learning based botnet detection using real-time connectivity graph based traffic features, 2014. US Patent 8,762,298.

[50] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proc. of LREC Workshop on New Challenges for NLP Frameworks*, 2010.

[51] William Robertson, Giovanni Vigna, Christopher Krügel, and Richard Kemmerer. Using generalization and characterization techniques in the anomaly-based detection of web attacks. In *Proc. of NDSS*, 2006.

[52] Saiph Savage, Andres Monroy-Hernandez, and Tobias Höllerer. Botivist: Calling volunteers to action using online bots. In *Proc. of CSCW*, 2016.

[53] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN. *ACM Trans. Database Syst.*, 42(3), 2017.

[54] Chenghui Shi, Xiaogang Xu, Shouling Ji, Kai Bu, Jianhai Chen, Raheem Beyah, and Ting Wang. Adversarial CAPTCHAs. *CoRR abs/1901.01107*, 2019.

[55] Leon Sixt, Benjamin Wild, and Tim Landgraf. RenderGAN: Generating realistic labeled data. *Frontiers in Robotics and AI*, 5:66, 2018.

[56] Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *Proc. of CCS*, 2003.

[57] Nedim Srndic and Pavel Laskov. Detection of malicious PDF files based on hierarchical document structure. In *Proc. of NDSS*, 2013.

[58] Brett Stone-Gross, Thorsten Holz, Gianluca Stringhini, and Giovanni Vigna. The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns. In *Proc. of LEET*, 2011.

[59] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proc. of CCS*, 2013.

[60] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When does machine learning FAIL? Generalized transferability for evasion and poisoning attacks. In *Proc. of USENIX Security*, 2018.

[61] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proc. Of ICLR*, 2013.

[62] Kymie M.C. Tan and Roy A. Maxion. Why 6? Defining the operational limits of stide, an anomaly-based intrusion detector. In *Proc. of IEEE S&P*, 2002.

[63] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. Fast anomaly detection for streaming data. In *Proc. of IJCAI*, 2011.

[64] Kurt Thomas, Chris Grier, Dawn Song, and Vern Paxson. Suspended accounts in retrospect: An analysis of Twitter spam. In *Proc. of IMC*, 2011.

[65] Kurt Thomas, Damon McCoy, Chris Grier, Alek Kolcz, and Vern Paxson. Trafficking fraudulent accounts: The role of the underground market in Twitter spam and abuse. In *Proc. of USENIX Security*, 2013.

[66] Ke Tian, Steve T. K. Jan, Hang Hu, Danfeng Yao, and Gang Wang. Needle in a haystack: Tracking down elite phishing domains in the wild. In *Proc. of IMC*, 2018.

[67] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In *Proc. of EUROCRYPT*, 2003.

[68] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y Zhao. You are how you click: Clickstream analysis for sybil detection. In *Proc. of USENIX Security*, 2013.

[69] Shuhao Wang, Cancheng Liu, Xiang Guo, Hongtao Qu, and Wei Xu. Session-based fraud detection in online E-commerce transactions using recurrent neural networks. In *Proc. of ECML-PKDD*, 2017.

[70] Guixin Ye, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, and Zheng Wang. Yet another text CAPTCHA solver: A generative adversarial network based approach. In *Proc. of CCS*, 2018.

[71] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient GAN-based anomaly detection. In *Proc. of ICLR Workshop*, 2018.

[72] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. In *Proc. of ICLR*, 2017.

[73] Panpan Zheng, Shuhan Yuan, Xintao Wu, Jun Li, and Aidong Lu. One-class adversarial nets for fraud detection. In *Proc. of AAAI*, 2019.

[74] Chong Zhou and Randy C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proc. of KDD*, 2017.

[75] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A C-LSTM neural network for text classification. *CoRR abs/1511.08630*, 2015.

[76] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proc. of ICCV*, 2017.

[77] Xinyue Zhu, Yifan Liu, Zengchang Qin, and Jiahong Li. Emotion classification with data augmentation using generative adversarial networks. In *Proc. of PAKDD*, 2018.

[78] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *Proc. of ICLR*, 2018.