2020 IEEE Symposium on Security and Privacy

Throwing Darts in the Dark? Detecting Bots with Limited Data using Neural Data Augmentation

Steve T.K. Jan[1,2], Qingying Hao[1], Tianrui Hu[2], Jiameng Pu[2],
Sonal Oswal[3], Gang Wang[1], Bimal Viswanath[2]

[1]University of Illinois at Urbana-Champaign [2]Virginia Tech [3]Radware
tekang@vt.edu, qhao2@illinois.edu, {tianruihu, jmpu}@vt.edu, SonalO@radware.com, gangw@illinois.edu, vbimal@vt.edu

***Abstract*—Machine learning has been widely applied to building security applications. However, many machine learning models require the continuous supply of representative labeled data for training, which limits the models' usefulness in practice. In this paper, we use bot detection as an example to explore the use of data synthesis 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 stream-based feature encoding scheme to support machine learning models for detecting advanced bots. The key novelty is that our model detects bots with extremely limited labeled data. We propose a data synthesis method to synthesize unseen (or future) bot behavior distributions. 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.**

I. Introduction

In recent years, machine learning (ML) has shown great success in building security defenses to protect computer and networked systems [1], [2], [3]. 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 such as malicious website detection [4], [5], malicious phone call classification [6], network traffic analysis [3], malware classification [7], [8], [9], [10], and intrusion detection [11], [2].

A common challenge faced by ML-driven systems is that they often require "labeled data" to train a good detection model [7], [5]. While it is already highly expensive to obtain labels (*e.g.*, via manual efforts), 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 [12].

To solve this problem, 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. For many of the security applications, however, 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 this paper, we use *bot detection* as an example to explore the use of data synthesis to enable bot detection with limited training data. We worked with a security company and 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. The "ground-truth" labels are provided by the security company's internal system — via a 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.

We argue that the true value of the machine learning model is to handle attacker behaviors that cannot be precisely expressed by "rules". As such, we excluded bots that were already precisely flagged by existing rules and focused on the remaining "advanced bots" that bypassed the rules. We proposed a novel feature encoding method to encode new traffic data as they arrive for stream-based bot detection. 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 space. Based

© 2020, Steve T.K. Jan. Under license to IEEE.

DOI 10.1109/SP40000.2020.00079

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. In addition, 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 of time). 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.

II. Background and Goals

A. Bot Detection

Bots are computer-controlled software that pretends to be real users to interact with online services and other users in online communities. While there are bots designed for good causes (search engine crawlers, research bots) [13], [14], [15], most bots are operated to engage malicious actions such as spam, scam, click fraud and data scrapping [16], [17], [1], [18], [19], [20], [21], [22]. While many existing efforts are devoted to bot detection, the problem is still challenging due to the dynamic-changing nature of bots.

Online Turing Tests. CAPTCHA is short for "Completely Automated Public Turning Test to tell Computers and Humans Apart" [23]. CAPTCHA is useful to detect bots but is limited in coverage. The reason is that aggressively delivering CAPTCHA to legitimate users would significantly hurt user experience. In practice, services want to deliver a minimum number of CAPTCHAs to benign users while maximizing the number of detected bots. As such, it is often used as a validation method, to verify if a suspicious user is truly a bot.

Rule-based Approaches. Rule-based detection approaches detect bots following predefined rules [24]. 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.

Machine Learning based Approaches. Machine learning techniques have been proposed to improve the detection performance [25], [26], [27]. A common way is supervised training with labeled bot data and benign user data [28], [29], [21], [22]. There are also unsupervised methods [30], [31], [32], but they are often limited in accuracy compared to supervised methods.

B. Challenges in Practice

There are various challenges to deploy existing bot detection methods in practice. In this work, we collaborate with a security company `Radware` to explore new solutions.

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) [21], [19], [29], [22]. 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.

Our Goals. With these challenges in mind, we build a machine learning model that works complementary to the existing rule-based system and the CAPTCHA system. The model is designed to be *generic*, which only relies on basic network-level information without taking any application-level information. We design an encoding scheme that allows the system to work on *anonymized* datasets, further improving its portability across web services. In addition, the system is stream-based, which processes incoming network traffic and make decisions in near real-time. More importantly, we use this opportunity to explore the impact of "limited training data" on model performance. We explore the benefits and limitations of data synthesis methods in enhancing the model against attackers' dynamic changes.

III. Dataset and Problem Definition

Through our collaboration with `Radware`, 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.

TABLE I: 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 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.

A. Data Collection

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

Table I 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 Radware.

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.

B. 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.

TABLE II: 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%)

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 paper, we empirically set $T = 30$. We perform bot detection on each IP-sequence.

C. Ground-truth Labels

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

CAPTCHA Labels. Radware 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, Radware's 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 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 [33], [34], [35], [36]. However, even the most advanced attack [34] 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 [37]. To the best of our knowledge, the CAPTCHA system used by Radware is not among the known vulnerable ones. Indeed, the CAPTCHA system could still be bypassed by human-efforts-based CAPTCHA farms [38]. 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 Radware'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. Radware's 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 II. 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 Section IV-A.

D. 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 [38] 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).

IV. Basic Bot Detection System

In this section, 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 section, we use all the available training data to examine model performance. In the next section, we introduce a novel data synthesis method to detect bots with limited data (Section V).

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.

A. Phase I: Filtering Simple Bots

As discussed in Section III-C, Radware'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 VI). This requires Phase II to detect the advanced bots that bypassed the rules.

Table III shows the filtering results. 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[1]. 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).

B. 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 [43]. Word2Vec can be applied to process network traffic [44]: 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

[1] We set $T = 30$ because the sequence length T needs to be reasonably large to obtain meaningful patterns [39]. 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 [40], [41], [42] which are complementary to our goals.

TABLE III: Ground-truth data of IP-sequences.

Website	August 2018			January 2019			September 2019		
	Rule Matched (Simple bot)	CAPTCHA (Advanced bot)	Benign	Rule Matched (Simple bot)	CAPTCHA (Advanced bot)	Benign	Rule Matched (Simple bot)	CAPTCHA (Advanced bot)	Benign
A	42,487	6,117	15,390	30,178	4,245	10,393	8,974	15,820	12,664
B	23,346	2,677	48,578	10,434	2,794	26,922	18,298	9,979	37,446
C	50,394	19,113	32,613	-	-	-	-	-	-

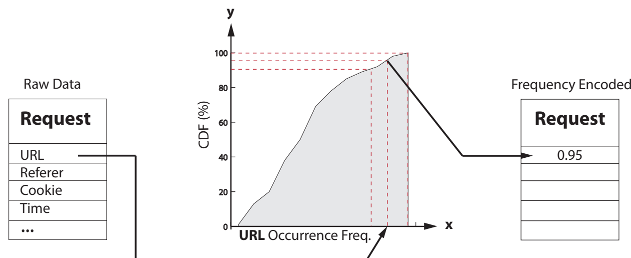


Fig. 1: Example of frequency encoding for the visited URL.

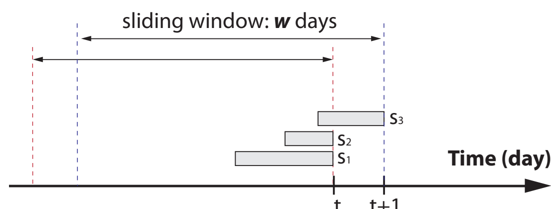


Fig. 2: Example of sliding window for feature encoding. S_1 and S_2 are IP sequences formed on day t, and S_3 is formed on day $t+1$. The feature vendors are encoded using the past w days of data.

to generate a high quality embedding [45], 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.

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 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 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.

TABLE IV: 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

Table IV 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 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 [44]. LSTM is a specialized Recurrent Neural Network (RNN) designed to capture the relationships of events in a sequence and is suitable to model sequential data [46], [47]. 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 [48], 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. We have tested other models such as Convolutional Neural Network (CNN), but LSTM performs better when training data is limited (Appendix A).

C. 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 [49] to ensure result validity.

TABLE V: 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 VI: 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

Training-Testing Temporal Split. We first ensure the *temporary training constraint* [49], 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. For our experiment, we have tested different window sizes w. We pick window size w =7 to balance the computation complexity and performance (additional experiments on window size are in Appendix B). The results for w =7 are shown in Table V.

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.

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 V, 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 VI 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% 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.

TABLE VII: 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

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 VII). 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).

V. Data Synthesis using ODDS

In this section, 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) [50], [51]. In our case, the main problem is the training data is too sparse to train an accurate model in the first place. We

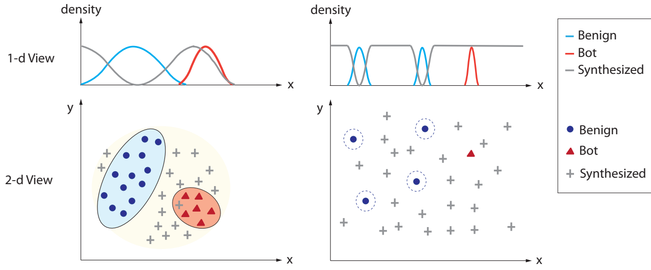


Fig. 3: Illustrating data synthesis in the clustered data region (left) and the outlier data region (right).

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.

A. 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) [52] 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 [53], because the input bot distribution is expected to be non-representative. 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. We need to synthesize data differently based on different internal structures of the labeled data. 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 outside of the benign clusters.

Figure 3 illustrates the high level idea of the data synthesis in clustered regions and outlier regions. In the following, we design a specialized GAN for such synthesis. We name the model "Outlier Distribution aware Data Synthesis" or ODDS.

B. 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 [54] on the 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, $\mathbf{M} = \{\mathbf{X}_1, \dots, \mathbf{X}_N\}$ is a labeled training dataset where $\mathbf{X_i}$ is an IP-sequence. $\mathbf{X_i} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$ 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 $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N\}$, 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 [54] 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 [55]. 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 $\mathcal{B}_o$.

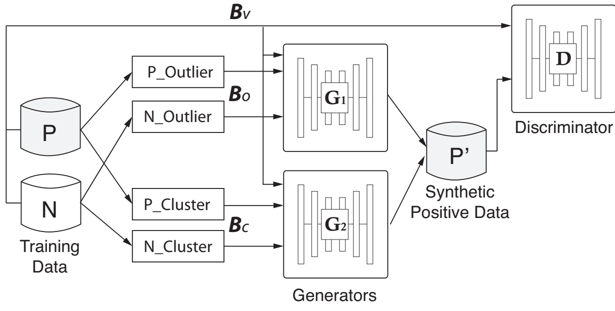


Fig. 4: The data flow of ODDS model. "Positive (P)" represents bot data; "Negative (N)" represents benign data.

C. Formulation of ODDS

As shown in Figure 4, 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.

Generator 1 for Outliers. To approximate the real outliers distribution p_{G_1}, the generator G_1 learns a generative 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 3 (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 p_{G_2}, the generator G_2 learns a generative 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:

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

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 (see Appendix C). In addition, using the discriminator for bot detection also eliminates the extra overhead of training a separate classifier.

TABLE VIII: 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	FP Rate	Precision	Recall	F1	FP Rate	Precision	Recall	F1	FP Rate
RF 100%	0.896	0.933	0.914	0.045	0.831	0.594	0.695	0.008	0.795	0.669	0.727	0.068
OCAN 100%	0.891	0.935	0.912	0.047	0.659	0.882	0.732	0.028	0.878	0.543	0.671	0.030
LSTM (ours) 100%	0.880	0.952	0.915	0.055	0.888	0.877	0.883	0.008	0.789	0.730	0.759	0.082
ODDS (ours) 100%	0.897	0.940	**0.918**	0.047	0.900	0.914	**0.902**	0.007	0.832	0.808	**0.815**	0.070
RF 1%	0.877	0.836	0.856	0.048	0.883	0.202	0.343	0.009	0.667	0.636	0.651	0.132
OCAN 1%	0.855	0.951	0.901	0.066	0.680	0.736	0.707	0.022	0.650	0.344	0.450	0.074
LSTM (ours) 1%	0.866	0.946	0.904	0.062	0.601	0.355	0.446	0.009	0.694	0.701	0.697	0.135
ODDS (ours) 1%	0.859	0.943	0.900	0.063	0.729	0.845	**0.783**	0.021	0.721	0.748	**0.734**	0.121

Implementation. To optimize the objective function of generators, we adapt several approximations. To minimize $\mathcal{H}(p_G)$, we adopt the pull-away term [56], [57]. To estimate p_m and p_b, we adopt the approach proposed by [58] which uses a neural network classifier to approximate.

VI. Performance 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?

A. 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 Section IV) 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.

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. More experiments on hyperparameters are in Appendix D.

Comparison Baselines. We evaluate our ODDS model with a series of baselines, including our basic LSTM model described in Section IV, and a non-deep learning model Random Forest [59]. We also include an anomaly detection method as a baseline. We select a GAN-based method called OCAN [60] 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. We have additional validation experiments in Appendix E, which shows OCAN indeed performs better than other traditional methods such as One-class SVM.

B. Training with 100% Training Data

Q1: *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 5 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 VIII (the upper half), presents a more detailed break up of performance into precision, recall, and false positive rate (*i.e.*, the fraction of benign users that are falsely classified as bots). We did not present the false negative rate since it is simply $1 - Recall$. The absolute numbers of false positives and false negatives are in Appendix F. 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.

C. Training with Limited Data

Q2: *How much does ODDS help when training with limited training data?*

As briefly shown in Section IV-C, 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.

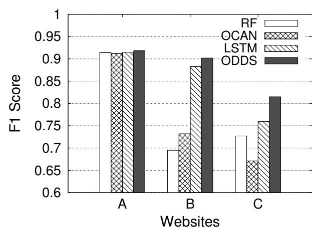


Fig. 5: Training with 100% training data in August 2018.

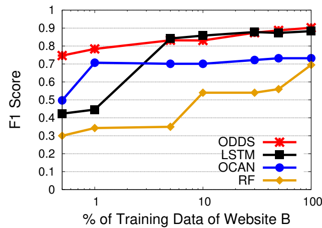
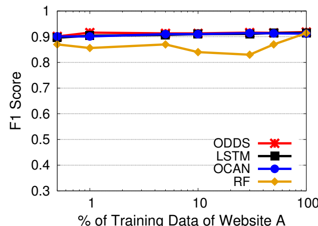
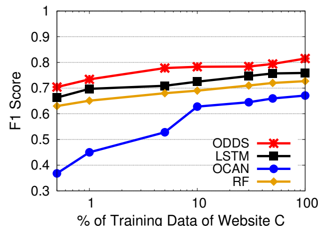


Fig. 6: Training with x% of training data in August 2018 (B).



(a) Website A



(b) Website C

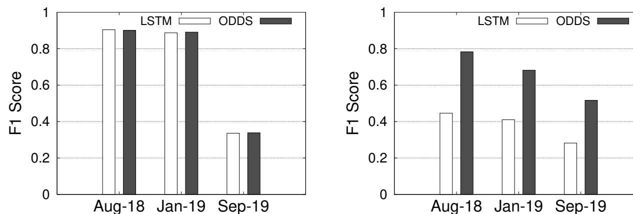
Fig. 7: Training with x% of training data in August-18 (A and C).

Figure 6 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 VIII, the performance gain is mostly coming from "recall", indicating the synthesized data is helpful to detect bots in the unknown distribution.

Figure 7 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 IX. 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

TABLE IX: 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



(a) Website A (b) Website B

Fig. 8: 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.

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.

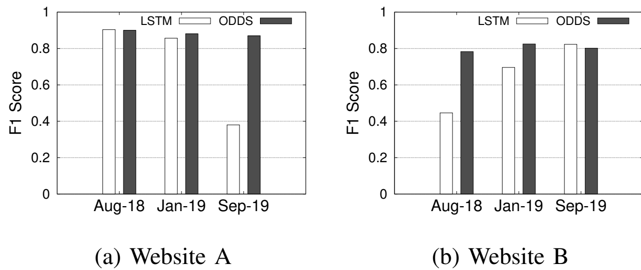
D. Generalizability in the Long Term

Q3: *Would ODDS help the classifier stay effective for a long period of time?*

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 8. 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.

Q4: *Does ODDS help with classifier re-training?*

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



(a) Website A (b) Website B

Fig. 9: 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.

TABLE X: 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

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

E. Contribution of Generators

Q5: *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 X, 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.

F. Insights into ODDS: Why it Works and When it Fails?

Q6: *At what condition does ODDS offer little or no help?*

TABLE XI: 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

TABLE XII: 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%

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 XI, 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 XII. 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 XII, 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.

TABLE XIII: 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

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.

G. Adversarial Examples and Adversarial Retraining

Q7: *Can ODDS benefit from adversarial re-training?*

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 [50] 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 [61], 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 XIII 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 [50] 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.

VII. Discussion

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).

Implications for the 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 VIII, 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.

Adversarial Evasion and Poisoning. ODDS improves model training with limited data. However, it does not mean attackers cannot evade ODDS by changing their behaviors. In fact, in Section VI-F, we already show that ODDS could not detect bots that are deeply embedded in the benign region (*e.g.*, whose nearest neighbors are 100% benign). In Section VI-G, we show that attackers in theory could identify adversarial examples in the whitebox setting. That said, we argue that 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 [62]. 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. We leave "cost-aware" adversarial evasion to future work.

Another potential attack against ODDS is poisoning attack. In theory, adversaries may also inject mislabeled data to the training set to influence the model training. The practical challenge, however, is to get the injected data to be considered as part of the training data, which has a cost. For example, to inject bot data point with a "benign" label, attackers will need to pay human labors to actually solve CAPTCHAs. We leave the further study of this attack to future work.

Limitations. Our study has a few limitations. First, while ODDS is designed to be generic, we haven't tested it beyond bot detection applications. Our method relies on the assumption that benign data is relatively stable and representative. As future work, we plan to test the system on other applications, and explore new designs when the benign set is also highly dynamic (*e.g.* website updates may cause benign users changing behaviors). Second, 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. Third, due to limited data (three disconnected months), we could not fully evaluate the impact of the sliding window and model retraining over a continuous time space. Fourth, we make detection decisions on IP-sequences. In practice, 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. Finally, ODDS needs to be retrained from scratch when new bot examples become available. A future direction of improvement is to perform incremental online learning [63] for model updation.

VIII. Related Work

Bot Detection. Bot detection is a well-studied area, and key related works have been summarized in Section II. Compared to most existing works on application-specific bots (*e.g.*, social network bots, game bots) [64], [21], [20], [29], [19], [65], [66], [67], we explicitly prioritize the model generalizability by avoiding any application or account specific features. Our main novelty is to explore the use of data synthesis for bot detection with limited data. We also show data synthesis helps to slow down the model decaying over time. One recent work [12] studied "concept drift" to determine *when* to retrain a classifier (for malware detection). Our work looks into a complementary direction by exploring ways to effectively retrain the classifier with limited data.

Anomaly Detection. Anomaly detection aims to detect anomalous data samples compared to known data distribution [68], [69], [70], [71], [72]. Researchers have applied anomaly detection methods to detect bots and other fraudulent activities [73], [60]. These works share a similar assumption with ODDS, that is, the normal/benign data should be (relatively) representative and stable. In our work, we use anomaly detection methods as our baselines, and show the benefit of synthesizing new data based on both the normal samples and the limited abnormal samples.

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 [74], image segmentation [75], image rendering [76], and emotion classification [77]. The most related work to ours is OCAN [60], which uses GAN to synthesize malicious samples for fraud detection. We have compared our system with OCAN in our evaluation, and demonstrated the benefits of using two generators to handle outliers and clustered data differently.

Recent works have explored introducing multiple generators to GAN [78], [79], [80], [81]. But their goals are to make the synthesized data (*e.g.*, synthesized images) closer to the target distribution. On the contrary, we are not interested in generating data that resemble the known bots, but to synthesize data for unknown bots. This calls for entirely different designs (*e.g.*, using different generators for outliers and clustered data).

IX. Conclusion

In this paper, we propose a stream-based bot detection model and augment it with a novel data synthesis method called ODDS. We evaluate our system on three different real-world online services. We show 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.

Acknowledgement

We thank our shepherd Suman Jana and anonymous reviewers for their constructive feedback. We also thank Harisankar Haridas for discussions on bot behavior. This work was supported by NSF grants CNS-1750101 and CNS-1717028.

REFERENCES

[1] V. Dave, S. Guha, and Y. Zhang, "ViceROI: Catching click-spam in search ad networks," in *Proc. of CCS*, 2013.

[2] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. of NDSS*, 2018.

[3] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *Proc. of USENIX Security*, 2016.

[4] G. Stringhini, C. Kruegel, and G. Vigna, "Shady paths: Leveraging surfing crowds to detect malicious web pages," in *Proc. of CCS*, 2013.

[5] K. Tian, S. T. K. Jan, H. Hu, D. Yao, and G. Wang, "Needle in a haystack: Tracking down elite phishing domains in the wild," in *Proc. of IMC*, 2018.

[6] H. Li, X. Xu, C. Liu, T. Ren, K. Wu, X. Cao, W. Zhang, Y. Yu, and D. Song, "A machine learning approach to prevent malicious calls over telephony networks," in *Proc. of IEEE S&P*, 2018.

[7] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of android malware in your pocket," in *Proc. of NDSS*, 2014.

[8] G. Dahl, J. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *Proc. of ICASSP*, 2013.

[9] N. Srndic and P. Laskov, "Detection of malicious PDF files based on hierarchical document structure," in *Proc. of NDSS*, 2013.

[10] S. E. Coull and C. Gardner, "Activation analysis of a byte-based deep neural network for malware classification," in *Proc. of DLS workshop*, 2019.

[11] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, "SAQL: A stream-based query system for real-time abnormal system behavior detection," in *Proc. of USENIX Security*, 2018.

[12] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *Proc. of USENIX Security*, 2017.

[13] T. Lokot and N. Diakopoulos, "News bots: Automating news and information dissemination on Twitter," *Digital Journalism*, vol. 4, pp. 682–699, 8 2016.

[14] S. Savage, A. Monroy-Hernandez, and T. Höllerer, "Botivist: Calling volunteers to action using online bots," in *Proc. of CSCW*, 2016.

[15] C. A. Davis, O. Varol, E. Ferrara, A. Flammini, and F. Menczer, "BotOrNot: A system to evaluate social bots," in *Proc. of WWW*, 2016.

[16] K. Chiang and L. Lloyd, "A case study of the rustock rootkit and spam bot," *HotBots*, vol. 7, no. 10-10, p. 7, 2007.

[17] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proc. of NDSS*, 2008.

[18] V. Dave, S. Guha, and Y. Zhang, "Measuring and fingerprinting click-spam in Ad networks," in *Proc. of SIGCOMM*, 2012.

[19] G. Wang, T. Konolige, C. Wilson, X. Wang, H. Zheng, and B. Y. Zhao, "You are how you click: Clickstream analysis for sybil detection," in *Proc. of USENIX Security*, 2013.

[20] K. Thomas, D. McCoy, C. Grier, A. Kolcz, and V. Paxson, "Trafficking fraudulent accounts: The role of the underground market in Twitter spam and abuse," in *Proc. of USENIX Security*, 2013.

[21] K. Thomas, C. Grier, D. Song, and V. Paxson, "Suspended accounts in retrospect: An analysis of Twitter spam," in *Proc. of IMC*, 2011.

[22] E. De Cristofaro, N. Kourtellis, I. Leontiadis, G. Stringhini, and S. Zhou, "LOBO: Evaluation of generalization deficiencies in Twitter bot classifiers," in *Proc. of ACSAC*, 2018.

[23] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *Proc. of EUROCRYPT*, 2003.

[24] R. Sommer and V. Paxson, "Enhancing byte-level network intrusion detection signatures with context," in *Proc. of CCS*, 2003.

[25] S. Kudugunta and E. Ferrara, "Deep neural networks for bot detection," *Information Sciences*, vol. 467, pp. 312–322, 2018.

[26] D. Damopoulos, S. A. Menesidou, G. Kambourakis, M. Papadaki, N. Clarke, and S. Gritzalis, "Evaluation of anomaly-based IDS for mobile devices using machine learning classifiers," *Security and Communication Networks*, vol. 5, no. 1, pp. 3–14, 2012.

[27] S. Ranjan, J. Robinson, and F. Chen, "Machine learning based botnet detection using real-time connectivity graph based traffic features," 2014, uS Patent 8,762,298.

[28] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. of BICT*, 2016.

[29] D. Freeman, S. Jain, M. Dürmuth, B. Biggio, and G. Giacinto, "Who are you? A statistical approach to measuring user authenticity." in *Proc. of NDSS*, 2016.

[30] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: A survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.

[31] L. Invernizzi, P. M. Comparetti, S. Benvenuti, C. Kruegel, M. Cova, and G. Vigna, "EVILSEED: A guided approach to finding malicious web pages," in *Proc. of IEEE S&P*, 2012.

[32] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proc. of KDD*, 2016.

[33] K. Bock, D. Patel, G. Hughey, and D. Levin, "unCaptcha: A low-resource defeat of reCaptcha's audio challenge," in *Proc. of WOOT*, 2017.

[34] G. Ye, Z. Tang, D. Fang, Z. Zhu, Y. Feng, P. Xu, X. Chen, and Z. Wang, "Yet another text CAPTCHA solver: A generative adversarial network based approach," in *Proc. of CCS*, 2018.

[35] W. Aiken and H. Kim, "POSTER: DeepCRACk: Using deep learning to automatically crack audio CAPTCHAs," in *Proc. of ASIACCS*, 2018.

[36] M. Mohamed, N. Sachdeva, M. Georgescu, S. Gao, N. Saxena, C. Zhang, P. Kumaraguru, P. C. van Oorschot, and W.-B. Chen, "A three-way investigation of a game-CAPTCHA: Automated attacks, relay attacks and usability," in *Proc. of ASIACCS*, 2014.

[37] C. Shi, X. Xu, S. Ji, K. Bu, J. Chen, R. Beyah, and T. Wang, "Adversarial CAPTCHAs," *CoRR abs/1901.01107*, 2019.

[38] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage, "Re: CAPTCHAs: Understanding CAPTCHA-solving services in an economic context," in *Proc. of USENIX Security*, 2010.

[39] K. M. Tan and R. A. Maxion, "Why 6? Defining the operational limits of stide, an anomaly-based intrusion detector," in *Proc. of IEEE S&P*, 2002.

[40] A. Pathak, F. Qian, Y. Hu, Z. Mao, and S. Ranjan, "Botnet spam campaigns can be long lasting: Evidence, implications, and analysis," in *Proc. of SIGMETRICS*, 2009.

[41] S. Nilizadeh, H. Aghakhani, E. Gustafson, C. Kruegel, and G. Vigna, "Think outside the dataset: Finding fraudulent reviews using cross-dataset analysis," in *Proc. of WWW*, 2019.

[42] B. Stone-Gross, T. Holz, G. Stringhini, and G. Vigna, "The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns," in *Proc. of LEET*, 2011.

[43] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. of NeurIPS*, 2013.

[44] S. Wang, C. Liu, X. Guo, H. Qu, and W. Xu, "Session-based fraud detection in online E-commerce transactions using recurrent neural networks," in *Proc. of ECML-PKDD*, 2017.

[45] R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora," in *Proc. of LREC Workshop on New Challenges for NLP Frameworks*, 2010.

[46] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. of INTERSPEECH*, 2010.

[47] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[48] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir *et al.*, "Wide & deep learning for recommender systems," in *Proc. of DLRS*, 2016.

[49] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. of USENIX Security*, 2019.

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

[51] Y. Qin, N. Carlini, I. Goodfellow, G. Cottrell, and C. Raffel, "Imperceptible, robust, and targeted adversarial examples for automatic speech recognition," in *Proc. of ICML*, 2019.

[52] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. of NeurIPS*, 2014.

[53] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient GAN-based anomaly detection," in *Proc. of ICLR Workshop*, 2018.

[54] M. Ester, H.-P. Kriegel, J. Sander, and X. 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.
[55] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, “DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN,” *ACM Trans. Database Syst.*, vol. 42, no. 3, 2017.
[56] Z. Dai, Z. Yang, F. Yang, W. W. Cohen, and R. Salakhutdinov, “Good semi-supervised learning that requires a bad GAN,” in *Proc. of NeurIPS*, 2017.
[57] J. Zhao, M. Mathieu, and Y. LeCun, “Energy-based generative adversarial network,” in *Proc. of ICLR*, 2017.
[58] A. Niculescu-Mizil and R. Caruana, “Predicting good probabilities with supervised learning,” in *Proc. of ICML*, 2005.
[59] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324
[60] P. Zheng, S. Yuan, X. Wu, J. Li, and A. Lu, “One-class adversarial nets for fraud detection,” in *Proc. of AAAI*, 2019.
[61] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, “Transferability in machine learning: From phenomena to black-box attacks using adversarial samples,” *CoRR abs/1605.07277*, 2016.
[62] O. Suciu, R. Marginean, Y. Kaya, H. D. III, and T. Dumitras, “When does machine learning FAIL? Generalized transferability for evasion and poisoning attacks,” in *Proc. of USENIX Security*, 2018.
[63] M. Du, Z. Chen, C. Liu, R. Oak, and D. Song, “Lifelong anomaly detection through unlearning,” in *Proc. of CCS*, 2019.
[64] D. M. Freeman, “Can you spot the fakes? On the limitations of user feedback in online social networks,” in *Proc. of WWW*, 2017.
[65] G. Wang, T. Wang, H. Zheng, and B. Y. Zhao, “Man vs. Machine: Practical adversarial detection of malicious crowdsourcing workers,” in *Proc. of USENIX Security*, 2014.
[66] H. Zheng, M. Xue, H. Lu, S. Hao, H. Zhu, X. Liang, and K. W. Ross, “Smoke screener or straight shooter: Detecting elite sybil attacks in user-review social networks,” in *Proc. of NDSS*, 2018.
[67] E. Lee, J. Woo, H. Kim, A. Mohaisen, and H. K. Kim, “You are a game bot! Uncovering game bots in MMORPGs via self-similarity in the wild,” in *Proc. of NDSS*, 2016.
[68] C. Zhou and R. C. Paffenroth, “Anomaly detection with robust deep autoencoders,” in *Proc. of KDD*, 2017.
[69] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, “Deep autoencoding gaussian mixture model for unsupervised anomaly detection,” in *Proc. of ICLR*, 2018.
[70] M. Amer, M. Goldstein, and S. Abdennadher, “Enhancing one-class support vector machines for unsupervised anomaly detection,” in *Proc. of KDD Workshop*, 2013.
[71] S. C. Tan, K. M. Ting, and T. F. Liu, “Fast anomaly detection for streaming data,” in *Proc. of IJCAI*, 2011.
[72] W. Robertson, G. Vigna, C. KrÃijgel, and R. Kemmerer, “Using generalization and characterization techniques in the anomaly-based detection of web attacks.” in *Proc. of NDSS*, 2006.
[73] G. Jacob, E. Kirda, C. Kruegel, and G. Vigna, “PUBCRAWL: Protecting users and businesses from crawlers,” in *Proc. of USENIX Security*, 2012.
[74] A. Antoniou, A. Storkey, and H. Edwards, “Data augmentation generative adversarial networks,” *CoRR abs/1711.04340*, 2017.
[75] C. Bowles, L. Chen, R. Guerrero, P. Bentley, R. Gunn, A. Hammers, D. A. Dickie, M. V. Hernández, J. Wardlaw, and D. Rueckert, “GAN augmentation: Augmenting training data using generative adversarial networks,” *CoRR abs/1810.10863*, 2018.
[76] L. Sixt, B. Wild, and T. Landgraf, “RenderGAN: Generating realistic labeled data,” *Frontiers in Robotics and AI*, vol. 5, p. 66, 2018.
[77] X. Zhu, Y. Liu, Z. Qin, and J. Li, “Emotion classification with data augmentation using generative adversarial networks,” in *Proc. of PAKDD*, 2018.
[78] Z. Yi, H. Zhang, P. Tan, and M. Gong, “DualGAN: Unsupervised dual learning for image-to-image translation,” in *Proc. of ICCV*, 2017.
[79] H. Tang, D. Xu, W. Wang, Y. Yan, and N. Sebe, “Dual generator generative adversarial networks for multi-domain image-to-image translation,” in *Proc. of ACCV*, 2018.
[80] S. Arora, R. Ge, Y. Liang, T. Ma, and Y. Zhang, “Generalization and equilibrium in generative adversarial nets (GANs),” in *Proc. of ICML*, 2017.
[81] Q. Hoang, T. D. Nguyen, T. Le, and D. Phung, “MGAN: Training generative adversarial nets with multiple generators,” in *Proc. of ICLR*, 2018.
[82] L. M. Manevitz and M. Yousef, “One-class SVMs for document classification,” *Journal of Machine Learning Research*, 2002.

APPENDIX A: LSTM VS. CNN

To justify our choice of Long-Short-Term-Memory (LSTM) model [44], we show the comparison results with Convolutional Neural Network (CNN) using the same feature encoding methods on the same dataset. The architecture of the CNN model is a stack of two convolutional layers (with 64 filters and 32 filters), followed by one fully connected layer with a sigmoid activation function. We experiment with 1% as well as 100% of the data from Website B in August 2018 for training. As shown in Table XIV, the performance of CNN is not as high as LSTM under 1% training data. The performance is comparable under 100% of the training data. As we mentioned, our main contribution is the feature encoding method rather than the choice of deep neural networks. Our result shows that LSTM has a small advantage over CNN.

TABLE XIV: We use August-18 dataset from Website B; Models are trained with 1% of the training dataset.

	% of Data	Precision	Recall	F1
LSTM	1%	0.60	0.36	0.45
	100%	0.89	0.88	0.88
CNN	1%	0.62	0.29	0.37
	100%	0.85	0.93	0.89

APPENDIX B: IMPACT OF SLIDING WINDOW SIZES

The size of the sliding window (w) could affect the detection results. Our dataset (a month worth of data) does not allow us to test big window sizes. As such, we test the window size of 3 days, 5 days, and 7 days and present the results in Table XV. The results show that the window size of 7 gives better results than 3 and 5 when using 1% of the training data. 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). This trend was observed when using 100% of the training data, as shown in Table XV.

TABLE XV: Results of using different sliding window sizes (in days). We use August-18 dataset from Website B; Models are trained with 1% or 100% of the training dataset.

	Window Size	Precision	Recall	F1
1% of Data	3	0.585	0.331	0.422
	5	0.600	0.314	0.412
	7	0.601	0.355	0.446
100% of Data	3	0.912	0.915	0.913
	5	0.937	0.889	0.910
	7	0.888	0.877	0.883

Appendix C: Feeding Synthetic Data To Other Classifiers

The discriminator of ODDS can be directly used for bot detection. A natural follow-up question is, what if we feed the synthetic data generated by ODDS to other classifiers? Can we improve the performance of the original classifiers? How is the performance compared with using the discriminator? To answer these questions, we feed the synthetic data to our LSTM model, and a traditional method, Random Forest (RF). We generate 600 synthetic data points based on the 1% of bot training samples in the August 2018 dataset.

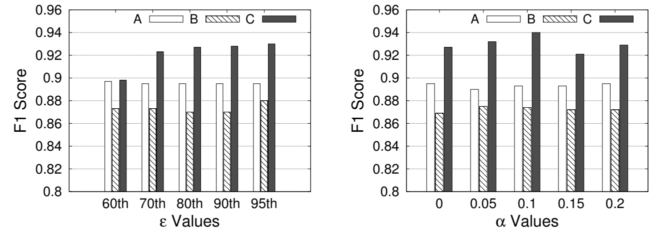
TABLE XVI: Feeding synthetic data to LSTM and RF. We use August-18 dataset from Website B; Models are trained with 1% of the training dataset.

	Synthetic data?	Precision	Recall	F1
RF	No	0.883	0.202	0.343
	Yes	0.826	0.570	0.596
LSTM	No	0.601	0.355	0.446
	Yes	0.698	0.757	0.719
ODDS	Yes	0.729	0.845	0.783

As shown in Table XVI, by feeding synthetic data to the classifier training, both models' performance is improved. The F1 score of LSTM is improved from 0.446 to 0.719, and the F1 score of RF is improved from 0.343 to 0.596. Despite the performance improvements, the LSTM model and the RF model are still not as accurate as the discriminator of ODDS. One possible explanation is that the synthetic data is generated in the latent space. To feed the data to other classifiers, we need to use the decoder to convert the latent vectors back to the original feature space, which may introduce some distortions during the reconstruction. In our paper, the discriminator is a better choice for bot detection also because it eliminates the need/overhead for training a separate classifier.

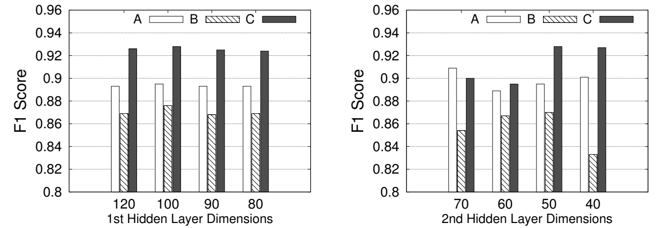
Appendix D: HyperParameters of ODDS

We have examined the model performance with respect to different hyperparameter settings for ODDS. The methodology is to split the training dataset into a training set and a validation set, and use the validation set to tune the parameters. For example, we train the model using 80% of first two weeks of August 2018, and use 20% of the data as the validation set to justify our parameters setting. We fix all the parameters to the default setting, and then examine the validation result by changing one parameter at a time. Figures 10a shows the validation results for different ϵ values. ϵ is the threshold for ODDS's generators (both G1 and G2) to determine if the generated bot samples are in the high-density regions of benign users). We set ϵ to the K_{th} percentile of real benign users' distribution. Figure 10b shows different α. α is the term for G2 to control how close the synthesized bot samples are to real bot samples and to real benign samples. For website A and website B, their validation performance is not too sensitive to α and ϵ. For website C, $\alpha = 0.1$ can achieve the highest validation performance. Note that τ_1, τ_2 and C in our equations can be omitted because both terms are constant, and the gradients with respect to these terms are mostly zero.



(a) F1 score for different ϵ (b) F1 score for different α

Fig. 10: F1 score on the *validation set* for different ϵ and α for website A, B, C.



(a) F1 score for different dimensions of first layer (b) F1 score for different dimensions of second layer

Fig. 11: F1 scores on the *validation set* using different dimensions for layers in the generators and the discriminator for website A, B, C.

Figure 11 shows the validation performance for different dimensions of the first and second hidden layers of the discriminator and generator. These results suggest setting 100 and 50 dimensions for the first and the second layers lead to a good validation performance.

TABLE XVII: Characterizing different datasets (August 2018).

Website	Avg. Distance Between Train and Test (benign)	Avg. Distance Between Train and Test (bots)
A	0.291	0.237
B	0.233	0.358
C	0.303	0.313

We notice that website C has the best validation F1 score in the different settings above. This is different from the main results on the *testing set* where C has the lowest F1 score (see Figure 5). We suspect that C's testing data is very different from the training data, which could explain why C has the best validation result but has the worst testing result. Table XVII shows some statistics to support this hypothesis. We compute the average distance between the training and testing samples, for the bots and benign users separately. We notice that C has a high distance between training and testing set, especially for the benign users. It is possible that concept drift happened even during a short time span such as within a month. Such discrepancies between the training and the testing data could hurt the testing performance.

TABLE XVIII: 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		
	FN rate	FN	FP	FN rate	FN	FP	FN rate	FN	FP
RF 100%	0.069	221	342	0.386	767	244	0.345	2898	1370
OCAN 100%	0.065	209	363	0.192	383	820	0.454	3814	666
LSTM (ours) 100%	0.048	152	416	0.123	245	219	0.270	2264	1632
ODDS (ours) 100%	0.059	190	360	0.086	171	200	0.199	1670	1401
RF 1%	0.185	593	364	0.703	1396	254	0.365	3067	2625
OCAN 1%	0.049	157	503	0.283	564	632	0.670	5622	1486
LSTM (ours) 1%	0.054	173	471	0.611	1214	261	0.294	2467	2685
ODDS (ours) 1%	0.056	181	481	0.158	314	615	0.253	2128	2411

APPENDIX E: ONE-CLASS SVM.

Our method and other anomaly detection methods share a similar assumption that the benign data is relatively more stable. In our paper, we selected OCAN, a recently published anomaly detection method, as the comparison baseline. Here we show the results of another popular anomaly detection method called One-class SVM [82]. One-class SVM aims to separate one class of samples from all the others by constructing a hyper-plane around the data samples. In this experiment, we use "benign" data as the known class. As shown in Table XIX, this anomaly detection method does not perform well on our dataset. One-class SVM tends to a high recall but a very low precision. The performance is not as high as OCAN (and our method ODDS) in both settings (1% and 100% training data).

TABLE XIX: Evaluation of One-class SVM using August-18 dataset.

Website	% of Data	Precision	Recall	F1
A	1%	0.407	0.991	0.577
	100%	0.441	0.990	0.611
B	1%	0.110	0.988	0.193
	100%	0.09	0.990	0.197
C	1%	0.336	0.745	0.463
	100%	0.336	0.747	0.464

APPENDIX F: FALSE POSITIVES AND FALSE NEGATIVES

To complement the main results in Table VIII, we add a new Table XVIII to show the absolute numbers of false positives and false negatives as well as the false negative rate. False negative rate is the fraction of the true bots that are misclassified as benign. Combining the results in Table VIII, and Table XVIII, we show that our system ODDS can drastically increase the number of detected true bots (reducing false negative rate) while producing comparable number of false positives. In practice, these false positives can be further reduced by the CAPTCHA system (it affects user experience but at a reasonably small scale).