Robustifying Machine Learning based Security Applications

Steve T.K. Jan

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

Doctor of Philosophy
in
Computer Science and Applications

Gang Wang, Co-chair
Bimal Viswanath, Co-chair
Jia-Bin Huang
Naren Ramakrishnan
Danfeng (Daphne) Yao
Xinyu Xing

August 3, 2020
Blacksburg, Virginia

Keywords: Machine Learning, Security, Bot Detection, Phishing Attacks

Robustifying Machine Learning based Security Applications

Steve T.K. Jan

(ABSTRACT)

In recent years, machine learning (ML) has been explored and employed in many fields. However, there are growing concerns about the robustness of machine learning models. These concerns are further amplified in security-critical applications — attackers can manipulate the inputs (i.e., adversarial examples) to cause machine learning models to make a mistake, and it's very challenging to obtain a large amount of attackers' data. These make applying machine learning in security-critical applications difficult.

In this dissertation, we present several approaches to robustifying three machine learning based security applications. First, we start from adversarial examples in image recognition. We develop a method to generate robust adversarial examples that remain effective in the physical domain. Our core idea is to use an image-to-image translation network to simulate the digital-to-physical transformation process for generating robust adversarial examples. We further show these robust adversarial examples can improve the robustness of machine learning models by adversarial retraining. The second application is bot detection. We show that the performance of existing machine learning models is not effective if we only have the limit attackers' data. We develop a data synthesis method to address this problem. The key novelty is that our method is distribution aware synthesis, 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 show the detection performance using 1% of attackers' data is close to existing methods trained with 100% of the attackers' data. The third component of this dissertation is phishing detection. By designing a novel measurement system, we search and detect phishing websites that adopt evasion techniques not only at the page content level but also at the web domain level. The key novelty is that our system is built on the observation of the evasive behaviors of phishing pages in practice. We also study how existing browsers defenses against phishing websites that impersonate trusted entities at the web domain. Our results show existing browsers are not yet effective to detect them.

Robustifying Machine Learning based Security Applications

Steve T.K. Jan

(GENERAL AUDIENCE ABSTRACT)

Machine learning (ML) is computer algorithms that aim to identify hidden patterns from the data. In recent years, machine learning has been widely used in many fields. The range of them is broad, from natural language to autonomous driving. However, there are growing concerns about the robustness of machine learning models. And these concerns are further amplified in security-critical applications — Attackers can manipulate their inputs (i.e., adversarial examples) to cause machine learning models to predict wrong, and it's highly expensive and difficult to obtain a huge amount of attackers' data because attackers are rare compared to the normal users. These make applying machine learning in security-critical applications concerning.

In this dissertation, we seek to build better defenses in three types of machine learning based security applications. The first one is image recognition, by developing a method to generate realistic adversarial examples, the machine learning models are more robust for defending against adversarial examples by adversarial retraining. The second one is bot detection, we develop a data synthesis method to detect malicious bots when we only have the limit malicious bots data. For phishing websites, we implement a tool to detect domain name impersonation and detect phishing pages using dynamic and static analysis.

Acknowledgments

I give my deepest thanks to my advisor Gang Wang, who has mentored me from when I had very limited knowledge about security but thought it was cool and important. I am exceedingly grateful to Gang for always putting his time and energy first for me to discuss my projects. During my Ph.D. study, I stuck countless times at some points, but we are always able to discuss some possible solutions. His patience and cheerful attitude have saved me many times from self-doubt.

I thank Bimal Viswanath for discussing the bot detection project with me, and providing valuable suggestions and directions. I thank Jia-Bin Huang, I really enjoyed brainstorming on our adversarial examples project with you and also teach me how to do a good presentation. I thank Naren Ramakrishnan for collaborating with me on our CIKM paper. I am impressed by your leadership and deep knowledge of data mining. I thank Danfeng Yao for working with me on our phishing websites detection project. Your enthusiasm always motivates me. I also thank Xinyu Xing for discussing with me for the explanation project. This experience helped me a lot for my internship at IBM research.

Many thanks as well to both the current and past members in the labs: Hang Hu, Limin Yang, Qingying Hao, Chao Xu, Tianrui Hu, Peng Peng, Xiangwen Wang, Chun Wang. I also thank all of the friends in Computer Science department. I will miss eating lunching, gossiping around, and discussing crazy ideas with you guys. Studying Ph.D. is stressful, but those friends make the journey easier.

I am deeply thankful to my family for their love, support, and sacrifices. Without them, this study would never have been finished. This last word of acknowledgment I have saved for my dear wife Helen Hsu, who has been with me all these years and has made them the best years of my life.

Contents

List of Figures ix

List of Tables xi

1 Introduction 1

1.1 Motivation . . . 1

1.2 Background . . . 2

1.3 Dissertation Outline . . . 3

1.3.1 Image Recognition: Improving Robustness of Models by Retraining with Realistic Adversarial Examples . . . 3

1.3.2 Bot Detection: Data Augmentation to Tackle Data Scarcity Problem 4

1.3.3 Phishing Websites Detection: Measurement-Driven Analysis to Understand Adaptive Attackers . . . 5

1.4 Dissertation Statement . . . 6

1.5 Research Contributions . . . 6

2 Related Works 8

2.1 Image Recognition . . . 8

2.2 Bot Detection . . . 9

2.3 Phishing Websites Detection . . . 10

3 Image Recognition: Improving Robustness of Models by Retraining with Realistic Adversarial Examples 12

3.1 Introduction . . . 12

3.2 Generating Adversarial Examples . . . 13

3.2.1 Basic Iterative Method (BIM) . . . 14

3.2.2 Expectation over Transformation (EOT) . . . 14

3.2.3 Robust Physical Perturbations (RP_2) . . . 15
3.3 Our Method . . . 15
3.3.1 Step 1: Training An Image-to-image Translation Network . . . 16
3.3.2 Step 2: Apply Exception over Transformation (EOT) . . . 17
3.3.3 Step 3: Add Noise to Simulated Physical Image . . . 17
3.4 Experimental Evaluation . . . 18
3.4.1 Experiment Setups . . . 18
3.4.2 Experiment Process . . . 20
3.4.3 Exp A: Effectiveness of Adversarial Examples . . . 21
3.4.4 Exp B: Robustness against Viewing Angles . . . 22
3.4.5 Exp C: Transferability of Adversarial Examples . . . 24
3.4.6 Exp D: Retraining using D2P . . . 25
3.5 Discussion and Conclusion . . . 25

4 Bot Detection: Data Augmentation to Tackle Data Scarcity Problem 27
4.1 Introduction . . . 27
4.2 Background and Goals . . . 28
4.2.1 Bot Detection . . . 28
4.2.2 Challenges in Practice . . . 29
4.3 Dataset and Problem Definition . . . 30
4.3.1 Data Collection . . . 30
4.3.2 Reprocessing: IP-Sequence . . . 31
4.3.3 Ground-truth Labels . . . 31
4.3.4 Problem Definition . . . 33
4.4 Basic Bot Detection System . . . 33
4.4.1 Phase I: Filtering Simple Bots . . . 34
4.4.2 Phase II: Machine Learning Model . . . 34
4.4.3 Evaluating The Performance . . . 37
4.5 Data Synthesis using ODDS . . . 39

4.5.1 Motivation of ODDS . . . 39
4.5.2 Overview of the Design of ODDS . . . 40
4.5.3 Formulation of ODDS . . . 41
4.6 Performance Evaluation . . . 44
4.6.1 Experiment Setup . . . 45
4.6.2 Training with 100% Training Data . . . 45
4.6.3 Training with Limited Data . . . 47
4.6.4 Generalizability in the Long Term . . . 48
4.6.5 Contribution of Generators . . . 50
4.6.6 Insights into ODDS: Why it Works and When it Fails? . . . 50
4.6.7 Adversarial Examples and Adversarial Retraining . . . 51
4.7 Discussion . . . 53
4.8 Conclusion . . . 54

5 Phishing Websites Detection: Measurement-Driven Analysis to Understand Adaptive Attackers 55

5.1 Introduction . . . 55
5.2 Background . . . 57
5.3 Detecting Phishing Pages Performing Evasive Page Contents . . . 58
5.3.1 Collect Ground Truth Phishing Pages . . . 58
5.3.2 Content Level Evasion Measurement . . . 59
5.3.3 Feature Engineering And Building A ML Classifier . . . 60
5.3.4 Ground-Truth Evaluation . . . 63
5.4 How Browsers Defending Against Phishing Pages Performing Evasive Domain Names . . . 64
5.4.1 Browsers' IDN Policies . . . 65
5.4.2 Building Testing Cases . . . 67
5.5 Browsers Measurement Methods and Results . . . 69
5.5.1 Testing Platform and Methods . . . 69

5.5.2 Results: Web Browsers . . . 71
5.5.3 Results: Mobile Browsers . . . 73
5.5.4 Browser Policy Changes Over Time . . . 74
5.5.5 Browsers vs. Real-world IDNs . . . 74
5.6 User Study For IDN Homograph Attacks . . . 76
5.6.1 Study Design . . . 76
5.6.2 Main User Study . . . 77
5.6.3 Overall Results . . . 79
5.6.4 Regression Analysis . . . 80
5.7 Discussion . . . 83
5.8 Conclusion . . . 84

6 Conclusion **85**
6.1 Summary . . . 85
6.2 Lessons Learned . . . 85
6.3 Implications for Our Works . . . 86
6.4 Future Research Directions . . . 87

Bibliography **89**

List of Figures

3.1 Adversarial examples are transformed across the digital and physical worlds before they enter the DNN image classifier. In practice, attackers have no (limited) control over the internal system. 14

3.2 We use a conditional Generative Adversarial Network (pix2pix or cycleGAN) to learn the digital-physical transformation for generating a synthetic physical image, which then serves as the "base" for producing adversarial noises. . . . 16

3.3 Adversarial examples in the physical domain. The targeted attack aims to make the classifier mis-classify the input image from the original label "*king penguin*" to the target label "*kite*". 18

3.4 Exp setups. 21

3.5 Img Printout. 21

3.6 Top-1 accuracy of target label for the adversarial examples under different viewing angles. 23

4.1 Example of frequency encoding for the visited URL. 34

4.2 Example of sliding window for feature encoding. S_1 and S_2 are IP sequences formed on day t, and S_3 is formed on day $t+1$. The feature vendors are encoded using the past w days of data. 35

4.3 Illustrating data synthesis in the clustered data region (left) and the outlier data region (right). 40

4.4 The data flow of ODDS model. "Positive (P)" represents bot data; "Negative (N)" represents benign data. 41

4.5 Training with 100% training data in August 2018. 46

4.6 Training with x% of training data in August 2018 (B). 46

4.7 Training with x% of training data in August-18 (A and C). 46

4.8 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. 48

4.9 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. 49

5.1 An example of page layout obfuscation of phishing pages (paypal). 58

5.2 Failure rates of the 10 testing categories for the latest version of four browsers. 70

5.3 Failure rates over time for different browser versions from January 2015 to April 2020, left figure is testing categories 1–6, and right figure is testing categories 7–10. 73

5.4 An example screenshot, which always shows the real webpage. The address bar was artificially added to display either the real domain name or a homograph IDN in Unicode (in this case, www.bankofamerḷca.com). Right below the screenshot, we asked *"Is the domain name in the browser address bar* bankofamerica.com*?"* Participants can choose one of three answers: *"Yes," "No," "I can't tell."* 77

5.5 Cumulative distribution function (CDF) of labeling accuracy for each user. . 79

5.6 The percentages of correct answers for different groups. We include the factors that have statistical significance in Table 5.12. "PNTA" under computing background stands for "Prefer not to answer." 80

6.1 F1 score on the *validation set* for different ϵ and α for website A, B, C. . . . 108

6.2 F1 scores on the *validation set* using different dimensions for layers in the generators and the discriminator for website A, B, C. 109

6.3 "googīe.com" in Gmail web interface. 113

List of Tables

3.1 Similarity (SSIM) and Dis-similarity (MSE) in comparison with the original digital image, after different several times of digital-to-physical transformation. 17

3.2 Similarity between the real physical image and the simulated physical image using pix2pix. 19

3.3 Classification confidence and accuracy of adversarial examples. BIM 1's noise level ($\epsilon = 30, \alpha = 0.5$) is the same with all other methods. BIM 2 uses bigger noises ($\epsilon = 70, \alpha = 0.5$). 19

3.4 Distribution of transformations t, where each parameter is sampled uniformly from the specified range. 20

3.5 Transferability of adversarial examples. "Base" represents the result of the original configuration of Exp. A and B. We then examine the performance of adversarial examples under different phone cameras, printers, and classifiers. 23

3.6 Statistics about before adversarial retraining and after adversarial retraining. 24

4.1 Dataset summary. 30

4.2 Estimated false positives of rules on IP-sequences. 32

4.3 Ground-truth data of IP-sequences. 33

4.4 Summaries of features and their encoding scheme. 35

4.5 The detection results of LSTM model on "advanced bots". 37

4.6 The overall detection performance The "precision" and "recall" are calculated based on all bots in August 2018 (simple bots and advanced bots). 37

4.7 F1-score when training with limited 1% of the labeled data of the August 2018 dataset. 38

4.8 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. 43

4.9 Characterizing different website datasets (August 2018). 47

4.10 F1 score when using only one generator; training with 100% of the training dataset of August 2018. 49

4.11 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. 50

4.12 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. 50

4.13 Applying adversarial training on LSTM and ODDS. We use August-18 dataset from website B; Models are trained with 1% of the training dataset. 52

4.14 Evaluation of One-class SVM using August-18 dataset. 53

5.1 String and code obfuscation in phishing pages. 60

5.2 Classifiers' performance on ground-truth data. 63

5.3 Detected squatting phishing pages by popular blacklists. VirusTotal contains 70+ blacklists. 64

5.4 Detected and confirmed squatting phishing pages. 64

5.5 The claimed policies of different browsers based on public documentations. 66

5.6 Testing cases and their related browser policies (the list of browser policies is in Table 5.5). 66

5.7 Tested browsers and their versions. 69

5.8 Testing results of the latest browsers. In total, 9,119 IDNs are tested per browser. We report the number of IDNS displayed as Unicode (*i.e.*, IDNs that browsers fail to block). 71

5.9 Example homograph IDNs that can bypass Chrome's policies to be displayed in Unicode. 73

5.10 Analysis results of .com DNS zone file. 75

5.11 Correct answer rates in the main study (6,510 answers): 94.6%, 48.5%, 55.2% for real, IDN-Block, IDN-Pass. 79

5.12 Logistic regression results: using website and user factors to predict whether the authenticity of a website domain name was correctly labeled by a user. . 81

6.1 We use August-18 dataset from Website B; Models are trained with 1% of the training dataset. 106

6.2 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. 107

6.3 Feeding synthetic data to LSTM and RF. We use August-18 dataset from Website B; Models are trained with 1% of the training dataset. 107

6.4 Characterizing different datasets (August 2018). 109

6.5 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. 110

6.6 IDN policies in email services and messaging apps. 112

Chapter 1

Introduction

1.1 Motivation

Internet users today are connected more widely and ubiquitously than ever before. While users are benefiting from the ease of access to information, the increased connectivity also presents new security challenges and creates significant national risks. The number of cyber incidents on federal systems reported to the U.S. Department oF Homeland Security in 2016 increased more than ten times in 2006 [150].

To defend against these increasing attacks, machine learning (ML) has been explored and it is widely used in many security applications [26, 65, 83, 136]. Driven by empirical data, machine learning algorithms can identify hidden patterns that cannot be easily expressed by rules or signatures. However, there are growing concerns about the robustness of machine learning models [55, 77, 100, 220]. One of the concerns is the machine learning models are vulnerable to the adversarial examples [24]: inputs specifically designed by an adversary to cause the models to misclassify them. And the concerns of robustness are getting amplified in the security applications — Attackers constantly craft new types of attacks with an increased level of sophistication to evade the detections [111], and it is usually not clear about what potential evasion techniques used by attackers in practice. Another common challenge is machine learning models often require the representative labeled data for training [117], and it is usually hard to get representative and large amounts of 'labeled attackers data' in the security applications because they are rare compared to the benign users. These concerns and challenges limit applying machine learning models to security applications.

The goal of this dissertation is to tackle these concerns in machine learning based security applications. For each of the concerns, we use one security application to illustrate. These concerns and the security applications are summarized as following:

- Vulnerability to adversarial examples: Image recognition
- Attackers data scarcity: Bot detection
- Adaptive attackers: Phishing websites detection.

Before outlining the contributions presented in subsequent chapters towards robustifying

machine learning based security applications, we first provide the relevant background of these applications, and how they connect to the concerns.

1.2 Background

Image recognition. The recent advancement in machine learning contribute the growth of image recognition, it has improved the performance of image recognition by utilizing deeper and wider networks [185]. Image recognition applies to a larger variety of computer vision tasks, for example to object-detection [102], segmentation [209] and human pose estimation [195].

These successes also applied to some security-critical applications. For example, face recognition [57]. In nowadays, it is common to use faces to unlock a smartphone and it has been implemented at many airports around the world for security checking. Another example is self driving car system [105], it exploits machine learning models to identify traffic lights, trees, curbs, pedestrians, street signs and other parts of any given driving environment.

However, they are vulnerable to adversarial examples [24]. This concern is particularly escalated after recent deadly crashes of self-driving vehicles [67]. For image classifiers, it has been shown that adding small perturbations to the original input image (known as "adversarial examples") can force an image classifier to make mistakes [113, 123, 184], which can yield practical risks. For example, an image classifier used to recognize stop signs for self-driving cars may mistake the sign as a yield sign if adversarial perturbations were added to the image (that are imperceivable to humans).

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) [52, 122, 169], the attackers leverage malicious bots to perform a wide array of malicious activities, such as spam, scam, click fraud and data scrapping [38, 50, 51, 53, 76, 190, 191, 201].

To detect these types of malicious bots, different approaches have been proposed. machine learning, in particular the supervised learning, exhibited promising results: examples of activity of benign (human) users and bots, labeled as such, can be fed to machine learning algorithms; trained models are then used to classify unforeseen accounts. Other alternative approached include unsupervised learning and rule base detection.

Nevertheless, one of the challenges in bot detection is lacking data of bots. And this is amplified by the different levels of sophistication of bots. Some bots are very simple while some of the bots are more sophisticated. These sophisticated bots have the capability to evade the existing detection and make it even harder to collect their data.

Phishing Websites Detection. Phishing has been widely used by cybercriminals to

steal user credentials and breach large networks. Typically, attackers would impersonate a trusted entity to gain the victim's trust, luring the victim to reveal important information. Phishing pages often act as the landing pages of malicious URLs distributed by phishing emails [112], SMS [161], or social network messages [59]. The phishing pages usually contain a form to trick users to enter passwords or credit card information.

Machine learning also applied to detect phishing websites [208]. By combining the machine learning classifier and designing the handpicked features that were selected based on domain knowledge (e.g., keyword frequency in the HTML pages, page rank information), the suspicious phishing websites are detected [208].

However, today's phishing websites are constantly *evolving to evade the detection*. Existing phishing blocklists [5] is ineffective to detect zero-day phishing pages, and some studies [193] show existing phishing pages have adopted evasion techniques that are likely to render existing detection methods ineffective. In addition, phishing pages impersonate the domain names of trusted entities to deceive users via domain squatting techniques [87, 106, 141]. For example, attackers may use faċebook.com to impersonate facebook.com. In this dissertation, we refer them as squatting phishing websites.

1.3 Dissertation Outline

With the background from the previous section, we now present our contributions towards more robust machine learning based security applications. This dissertation is organized into three main chapters.

1.3.1 Image Recognition: Improving Robustness of Models by Retraining with Realistic Adversarial Examples

Problem. Although many adversarial defense algorithms have been proposed [21, 24, 55, 64, 81, 82, 128, 153, 159, 196, 203, 220], they focus on adversarial examples in "digital domain", without considering the physical constraints in practice. Adversarial examples can also happen in "physical domain" [22], which is more practical and realistic than digital domain, since attackers usually do not have the ability to directly manipulate data inside such systems. To defend against the adversarial examples in physical domain, one way is to use model retraining [167]. To do so, it requires a number of adversarial examples that are effective in the physical domain. In other words, the adversarial perturbations should survive in different transformations (for example, different viewing angles and distances) and still are able to cause machine learning models to make mistakes.

Recent studies show that the effectiveness of adversarial examples degrades significantly under the various physical conditions (*e.g.*, different viewing angles and distances). Initial

efforts have been investigated to improve the robustness of adversarial examples by either synthesizing the digital images to simulate the effect of rotation, scaling, and perspective changes [22, 174] or manually taking "physical-domain" photos from different viewpoints and distances for producing robust physical adversarial examples [62, 63]. However, it either requires lots of manual efforts, or it did not consider the transformation introduced by physical devices (e.g., cameras, printers), which significantly limits its performance. Therefore, is there a way to craft robust adversarial examples efficiently?

Solution [97]. In Chapter 3, we propose a new method D2P to generate robust and realistic adversarial examples that can survive in the physical world with less manual efforts. The core idea is to explicitly simulate the digital-to-physical transformation of the physical devices (*e.g.*, paper printing, non-linear camera response functions, sensor quantization, and noises) to translate a digital image to its physical version before generating adversarial noises. We introduce an image-to-image translation layer based on conditional Generative Adversarial Networks (cGAN) to simulate this process. We experimented with pix2pix [95] and cycleGAN [222] models to carry out the transformation and redesign the noise generation to improve the robustness of the adversarial examples. With these robustness adversarial examples, we show the machine learning models are more robust to defend against adversarial examples.

1.3.2 Bot Detection: Data Augmentation to Tackle Data Scarcity Problem

Problem. There are three main existing methods to detect malicious bots. The first one is CAPTCHA. CAPTCHA is short for "Completely Automated Public Turning Test to tell Computers and Humans Apart" [199]. CAPTCHA is useful to detect malicious 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.

The second one is rule-based approaches. Rule-based detection approaches detect malicious bots following predefined rules [176]. 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.

The third one is machine learning based approaches. Machine learning techniques have been proposed to improve the detection performance [49, 111, 160]. A common way is supervised training with labeled bot data and benign user data [53, 68, 99, 190]. For malicious 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, the performance of supervised learning is limited given the limited attackers' data. Is there a way to synthesis

more attackers' data?

Solution [98]. In Chapter 4, we explore the design of a new data synthesis method ODDs. 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 on these intuitions, we designed a customized Generative Adversarial Network (GAN) with two complementary generators to synthesize clustered and outlier data simultaneously.

1.3.3 Phishing Websites Detection: Measurement-Driven Analysis to Understand Adaptive Attackers

Problem. As phishing attacks become prevalent [4], various phishing detection methods have been proposed, ranging from URL blacklisting [26] to visual similarity based phishing detection [132] and website content-based classification [208]. Visual similarity-based phishing detection [132] aims to compare the original webpages of popular brands to suspicious pages to detect "impersonation". Machine learning based methods [208] rely on features extracted from the HTML source code, JavaScript, and the web URLs to flag phishing websites. As phishing attacks evolve, these existing detection methods are limited. What are the evasion techniques used in the phishing sites? How can we detect them using machine learning? And how are browsers doing against these phishing attacks?

Solution [193]. In Chapter 5, we design a novel measurement system to search and measure these evasive techniques used by phishing websites. Particularly, we observe there are two high levels of evasions. The first high level is at the web content level. The attackers often hide phishing related text in the source code or change the layout of the phishing pages. The second high level is at the domain name level. The attackers register the domain names that are impersonating to the target brand to deceive the users while evading existing machine learning classifiers based on URL properties [26, 40, 127].

For the evasion at the web content level, we observe it can be categorized as code obfuscation, string obfuscation, and layout obfuscation. We build a detection system based on these observations. To this end, we apply visual analysis and *optical character recognition* (OCR) to extract key visual features from the page screenshots (particularly the regions of the login form). The intuition is that no matter how attackers obfuscate the HTML content, the visual presentation of the page will still need to look legitimate to deceive users. For the evasion at domain names level, we seek to understand how the existing browsers defending

against these evasive domain names, in particular for the homograph IDN domain [86]. Researchers have analyzed real-world DNS records and found homograph IDNs created for phishing [39, 115, 121, 183]. To mitigate this risk, browsers have recently introduced defense policies. However, it is not yet well understood regarding how these policies are constructed and how effective they are. To this end, we present an empirical analysis of browser IDN policies, and a user study to understand user perception of homograph IDNs.

1.4 Dissertation Statement

Robustifying machine learning models for security applications by combining empirical measurements, adversarial testing, and data synthesis.

1.5 Research Contributions

The high level goal of this dissertation is to develop the robust machine learning, and apply machine learning to solve security applications. My research contributes in following facets:

Develop new learning algorithms:

- We develop **D2P**, a method to craft the realistic adversarial examples that can fool the state-of-the-art image classifiers [185] in the physical domain. (Chapter 3)
- We propose **ODDs** to synthesize unseen (or future) malicious bots' behavior by generative models. (Chapter 4)

Propose new feature engineering methods:

- We develop a stream based feature encoding scheme to encode new traffic data. It allows us do perform real-time analysis and run bot detection on anonymized network data. (Chapter 4)
- We discover the robust features based on the observations of attackers' evasions techniques. These features are hard for attackers to modify. (Chapter 5)

Identify new attacks incidences:

- We identify and confirm 1,175 squatting phishing websites (857 web pages, 908 mobile pages) and report them to Google safe browsing. (Chapter 5)

- By constructing more than 9,000 testing cases, we report the potential holes of existing browsers to the corresponding bug/security teams of Chrome, Safari and Firefox. (Chapter 5)

Chapter 2

Related Works

We list the related works to this dissertation in image recognition, bot detection and phishing websites detection.

2.1 Image Recognition

Digital Adversarial Examples. Research first shows that deep neural networks are vulnerable to adversarial examples [184]. Since then various adversarial example generation algorithms have been proposed [35, 42, 113, 142, 152]. Beyond image classification, adversarial examples have shown success in manipulating deep neural networks for object detection and semantic segmentation [66, 209], and reinforcement learning agent [92, 109, 120]. However, most existing works only focus on the *digital domain*, assuming attackers can directly feed the digital version of the adversarial images into a DNN. This assumption is unrealistic. Take self-driving cars for example, it's less likely for an attacker to compromise the operating system to manipulate the digital images taken by the car cameras. Instead, a more realistic assumption is that attackers can perturb physical objects (*e.g.*, a movie poster) outside of the car, which will be captured (digitalized) by the camera before being classified by the DNN.

Physical Adversarial Examples. More recently, researchers started to explore how well adversarial examples can survive in the physical world. Results show that adversarial examples, while they can survive under a well-controlled environment [113], would lose the effectiveness in the physical world where there are spatial constraints (angle and distance), fabrication errors, and resolution changes [62, 124]. To construct more robust adversarial examples, researchers have tried to increase the amount of adversarial noises [123], but the drawback is the perturbations become more perceptible. Brown et al. [31] develop a scene-independent patch to fool classifiers, which again makes the adversarial examples obviously different from the original image (easily recognized). Athalye et al. [22] propose to apply digital transformations on the original images while generating adversarial noises. These transformations aim to simulate the changes of image conditions such as the perspective, the brightness, and the image scale. Sitawarin et al. [174] extend this work to traffic sign classifications. Sharif et al. [172] print the adversarial examples to fool a facial authentication

system.

However, existing works have two main limitations. *First*, most existing works evaluate their methods on an extremely small testing set (*e.g.*, 1–5 different traffic signs) [62, 63, 123, 174], which raises concerns on the generalizability to more complex objects. The only larger scale evaluation [113] focuses on *non-targeted* attacks (an easy attack) and the results suggest that physical domain attacks are much weaker, echoing the need for new methods to handle the physical domain transformation. *Second*, existing methods often require taking a large number of physical images [62], which is another unrealistic burden to bear.

2.2 Bot Detection

Bot Detection. Bot detection is a well-studied area, and key related works have been summarized in Section 4.2. Compared to most existing works on application-specific bots (*e.g.*, social network bots, game bots) [68, 69, 116, 190, 191, 201, 202, 218], 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 [101] studied "concept drift" to determine *when* to re-train a classifier (for malware detection).

Anomaly Detection. Anomaly detection aims to detect anomalous data samples compared to known data distribution [16, 166, 188, 221, 224]. Researchers have applied anomaly detection methods to detect bots and other fraudulent activities [96, 219]. 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 [17], image segmentation [29], image rendering [175], and emotion classification [223]. The most related work to ours is OCAN [219], 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 [19, 84, 189, 214]. 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).

2.3 Phishing Websites Detection

Squatting Domains Identification. Previous works have studied different types of squatting techniques [13, 141, 186]. For example, More et al. [141] measured typo squatting by generating a list of plausible misspellings of popular domains. Nikiforakis et al. [147] measured the bit squatting by generating a single bit-flip for a valid domain. Holgers et al. [87] characterized homograph squatting through character substitutions. Kinti et al. [106] measured combo squatting by searching domain keywords from DNS records. In this dissertation, we focus on aggregating and improving existing squatting methods to search for squatting phishing attacks.

Phishing Webpage Detection. A plethora of research has focused on blacklisting or content-based detection methods. For example, PhishTank [5] leverages crowdsourcing to collect phishing URLs that Internet users encountered. PhishEye [78] proposed to use honeypots to monitor live phishing pages. Other detection methods are based on visual similarities [132, 207] or lexical URL properties [26, 40, 127] to detect phishing pages. For example, DeltaPhish [43] detects compromised websites by comparing the page structure similarities. Cantina and Cantina+ [208, 216] are based on the keyword frequency and page rank information. Marchal et al. [131] also use keyword frequency in the HTML pages. A recent system Meerkat [28] uses deep learning models to analyze visual elements in webpages to detect compromised websites. Our approach is different since we use OCR to extract the text from the screenshots rather than focusing on the visual elements. Note that researchers of [12, 58] used OCR to extract keywords and query search engines to match again the real sites. However, this design still assumes phishing sites are similar/identical to the target sites, which is not necessarily true given the big variances introduced by the evasion techniques. Instead, we focus on more generic keywords extracted from logos, login forms, and other input fields to model the "phishing" attempts, which turns out to be effective.

Phishing Emails and Hosting Servers. Phishing emails are used to distribute the phishing URLs. Attackers can impersonate trusted parties to send phishing emails via email spoofing [89, 91] or email header injection [157]. In addition to registering squatting domains, attackers can also compromise existing web servers to host the phishing pages [149].

IDN Homograph. A number of studies have looked into IDN homograph. Researchers find that many of the IDN registrations are opportunistic, the domain names are owned by domain squatters [115, 121], and many of them are used for phishing and abuse [60, 121]. Another related project shows that most users do not have the knowledge of internationalized domain names [39]. It helps explain why IDNs can be deceptive. Compared to prior work [39, 115, 121, 183], our novelty comes from the detailed analysis of browser-level defense, and the discovered weaknesses of current IDN policies.

Security Indicators on URLs. Researchers have examined how users perceive and react to different URL presentations in browsers under security contexts. Most studies have reported negative results. For example, a recent study shows HTTPS Extended Validation (EV) certificate has little impact on users' security behavior [192]. In addition, prior work shows that domain name highlighting has limited effectiveness in warning users about malicious URLs [54, 119]. A closely related project looks into how different URL obfuscation methods (including IND homograph) affect users' ability to judge the authenticity of URLs [164]. The overall results are consistent with ours, showing that users have difficulties to correctly recognize obfuscated URLs.

Chapter 3

Image Recognition: Improving Robustness of Models by Retraining with Realistic Adversarial Examples

3.1 Introduction

In this chapter, we seek to robustify the machine learning models for defending against adversarial examples by retraining [167] with realistic adversarial examples. Unfortunately, most existing works of adversarial examples are 'unrealistic', which is only effective in the "digital domain", without considering the physical constraints in practice. A common assumption is that attackers can directly feed the *digital* images into the target classifiers [113, 142, 152, 172, 184]. However, this assumption is unrealistic since attackers have limited control on how the target system (*e.g.*, self-driving cars, surveillance cameras) takes photos. The different viewing angles and the non-linear camera response functions may substantially reduce the impact of the adversarial perturbations. These unrealistic examples may robustify the machine learning models for defending against adversarial examples in the digital domain, but not in the physical domain.

More recently, researchers started to study the feasibility of 'realistic' adversarial examples in the physical domain by printing out the images and re-taking them using cameras [113]. However, two challenges remain un-addressed that limit the feasibility of physical-domain adversarial examples. *First*, most existing methods [62, 63, 123, 174] are evaluated with an extremely small set of testing cases (*e.g.*, 5 cases in [62]). This is largely due to the expensive manual efforts required to conduct physical-domain experiments. There is a lack of large-scale evaluation to *fairly* and *thoroughly* assess different methods under a common ground. Second, existing methods, especially those relying on image synthesis, did not consider the transformation introduced by physical devices (*e.g.*, cameras, printers), which significantly limits its performance.

We craft realistic adversarial examples to address these challenges. *First*, we propose a new method (called **D2P**) to generate robust adversarial examples that can survive in the physical world. The core idea is to explicitly simulate the digital-to-physical transformation of the physical devices (*e.g.*, paper printing, non-linear camera response functions, sensor

quantization, and noises) to translate a digital image to its physical version before generating adversarial noises. We introduce an image-to-image translation layer based on conditional Generative Adversarial Networks (cGAN) to simulate this process. We experimented with pix2pix [95] and cycleGAN [222] models to carry out the transformation and redesign the noise generation to improve the robustness of the adversarial examples. *Second*, we conduct a large-scale experiment in the physical domain to evaluate our D2P method and compare it with three other state-of-the-art methods *under the same settings*. Our experiment takes advantage of a programmable rotational table to take a large number of photos semi-automatically (3000+ physical-domain images). The experiment validates the effectiveness of adversarial examples in the physical domain and shows that our method compares favorably with existing approaches. Our method also achieves a higher level of robustness (under different viewing angles) and transferability (under different cameras, printers, and models). Finally, we show the robustness of models is improved by retraining with adversarial examples generated by D2P.

We make three key contributions:

- We design a novel method **D2P** to generate robust adversarial examples against deep neural networks, by explicitly modeling the digital-to-physical transformation.
- We evaluate **D2P** using "physical-domain" experiments. We show that our adversarial examples are not only effective at the frontal view, but have a higher level of robustness across different viewing angles, and transfer well under different physical devices.
- We conduct a large-scale physical-domain experiment (3000+ physical images taken by cameras) that allows us to assess several related methods under the same setting to provide insights into their strengths and weaknesses.

3.2 Generating Adversarial Examples

In this section, we introduce the key methods for generating adversarial examples, including those that focus on the digital domain and those that aim to create adversarial examples for the physical domain. Here, we first define the problem. Adversarial examples are images that are carefully crafted to cause mis-classifications at testing time. Given an input image $\mathbf{X}$, the attack method generates adversarial noises and adds them to $\mathbf{X}$ to create an adversarial example $\mathbf{X}^{\text{adv}}$. The goal is to use $\mathbf{X}^{\text{adv}}$ to cause a mis-classification while keeping the noise sufficiently small to avoid alerting human observers. We denote y as the label of $\mathbf{X}$ and y' as the target label that $\mathbf{X}^{\text{adv}}$ aims to acquire ($y' \neq y$, and $\mathbf{X} \neq \mathbf{X}^{\text{adv}}$). The image classifier $F : [-1, 1]^{h \times w \times 3} \rightarrow \mathbb{R}^K$ takes an image of height h and width w as input, and produces the output of a probability distribution over K classes. Denote $L(F(\mathbf{X}), y)$ as the loss function that calculates the distance between the model output $F(\mathbf{X})$ and the target label y.

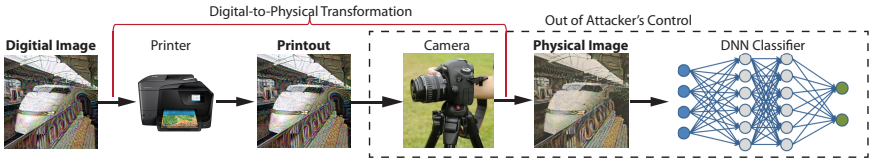


Figure 3.1: Adversarial examples are transformed across the digital and physical worlds before they enter the DNN image classifier. In practice, attackers have no (limited) control over the internal system.

3.2.1 Basic Iterative Method (BIM)

Basic iterative method presents a simple idea to generate adversarial noises [72]. The goal is to find a small δ so that $F(\mathbf{X}+\delta) = y'$. The method aims to solve the following objective function:

$$\arg\min_{\delta} \; L(F(\mathbf{X}+\delta), y') + c \cdot ||\delta||_p$$

where c controls the regularization of the distortion, and $||\delta||_p$ is the L_p norm that specifies $||\mathbf{X}^{\text{adv}} - \mathbf{X}||_p < \delta$. The optimization aims to cause a mis-classification from y to y' while minimizing the perturbation to x.

BIM *does not* consider the physical world challenges. As shown in Figure 3.1, it is unlikely that attackers can directly feed the generated adversarial example (a digital image) into the classifier. More practically, the digital image can be printed by the attacker as a physical object (*e.g.*, a poster), which is then captured by the camera of the target system (*e.g.*, a self-driving car) and digitalized into a new image (referred as "physical image"). This physical image is the actual input of the classifier. Since attackers have very limited control over the internal parts of the system, the different angles to take the photo or the nonlinear response functions of the camera can affect the attack success rate.

3.2.2 Expectation over Transformation (EOT)

The EOT method [22] aims to improve the robustness of adversarial examples using a series of synthetically transformed images (in the digital domain). More specifically, EOT applies a transformation function t to generate a distribution T for noise optimization, in order to make the perturbation δ more robust to physical changes. The objective function is of the form:

$$\arg\min_{\delta} \; \mathbb{E}_{t\in T} \; L(F(t(\mathbf{X}+\delta)), y') + c \cdot ||\delta||_p.$$

Here, transformation t can be either image translation, rotation, scaling, lighting variations, and contrast changes. Note that, however, *EOT* is solely based on the synthesis of *digital* images, which still ignores the physical effects introduced by the digital-to-physical transformations.

3.2.3 Robust Physical Perturbations ($\mathbf{RP}_2$)

The RP_2 method [62] enhances the EOT method by also considering *the physical images*. The RP_2 method, however, requires the attacker to print out a clean image and take a number of photos of the printout from different angles and distances (physical images). The set of physical images are denoted as $\mathbf{X}^V$. RP_2 solves this optimization:

$$\arg\min_{\delta} \ \mathbb{E}_{t\in T, x\in\mathbf{X}^V} \ L(F(t(x+\delta)), y') + c\cdot||\delta||_p$$

RP_2 is only tested on *5* road signs, and it is not yet clear if the method is broadly applicable; More importantly, the need of manually printing and taking multiple photos for producing *each* adversarial example hurts its practical value.

For all the methods above, the optimization problem can be solved by stochastic gradient descent and back-propagation, provided that the classifier F is differentiable. The expectation can be approximated by empirical mean (*i.e.,* Monte Carlo integration). For instance, in basic iterative method, $\mathbf{X}^{\text{adv}}$ is obtained when the following optimization equations 3.1 converge. Note that the "clip" function is to ensure that $\mathbf{X}^{\text{adv}}$ is a valid image and L_∞ ϵ-neighborhood of the clean image $\mathbf{X}$.

$$\begin{aligned}\mathbf{X}^{\text{adv}}_{N+1} &= \mathbf{X}^{\text{adv}}_{N} + \alpha\text{sign}(\bigtriangledown J(\mathbf{X}^{\text{adv}}_{N-1}, y')) \\ \mathbf{X}^{\text{adv}}_{N+1} &= \text{clip}(\mathbf{X}^{\text{adv}}_{N+1}, \mathbf{X}+\epsilon, \mathbf{X}-\epsilon)\end{aligned} \tag{3.1}$$

Defining Key Terms. We use Figure 3.1 to define important terms for the rest of the paper. (1) "*digital image*": the original image in the digital form. (2) "*printout*": the printed paper/poster of the original image. (3) "*physical image*": the photograph of the printout taken by a camera.

3.3 Our Method

In this section, we present a simple yet surprisingly effective method to generate robust adversarial examples in the physical world. The core idea is to explicitly simulate the physical-to-digital transformation introduced by (1) crafting the physical object (*e.g.*, image printing), and (2) digitalizing the physical object by the target system (*e.g.*, by a camera). Our goal is to generate adversarial noises that can survive the digital-to-physical transformation in practice. In addition, our method remains simulation-based, which eliminates the costly process of manually taking *physical images* for every single adversarial example (unlike RP_2). We call our method **D2P**, short for "digital-to-physical transformation". Figure 3.2 shows the high-level workflow.

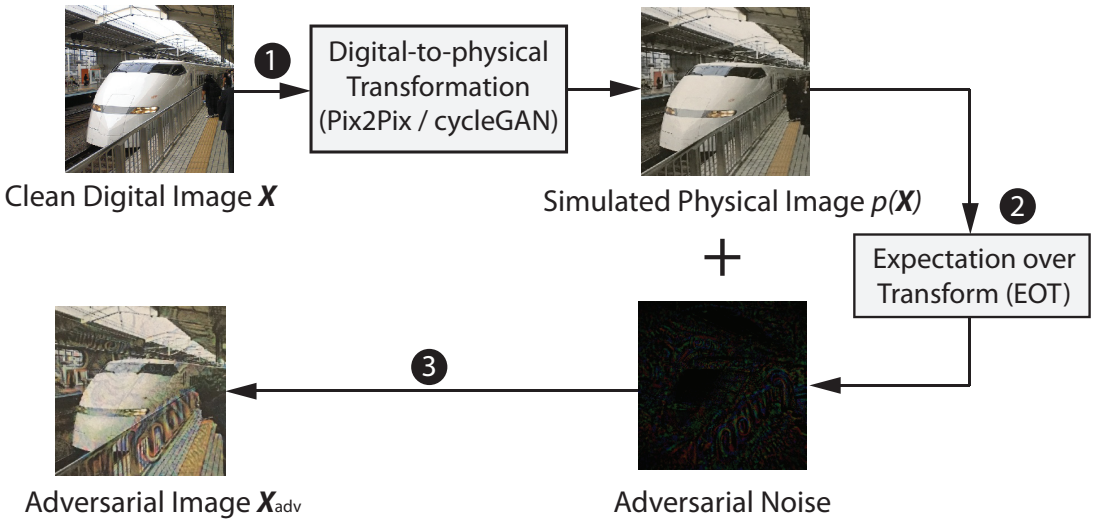


Figure 3.2: We use a conditional Generative Adversarial Network (pix2pix or cycleGAN) to learn the digital-physical transformation for generating a synthetic physical image, which then serves as the "base" for producing adversarial noises.

3.3.1 Step 1: Training An Image-to-image Translation Network

We first simulate the digital-to-physical transformation using a conditional Generative Adversarial Networks (cGANs) for performing image-to-image translation [95, 222].[1] The cGANs model has shown successes in tasks such as labeling maps and coloring images. We tailor a network to capture the transformation from a digital image to its physical version to simulate the nonlinear quantization effect of physical devices (*e.g.*, cameras).

Our cGANs model is trained to learn mapping function $p : D \rightarrow P$ where D is a set of images in the digital domain and P is a set of physical images (*i.e.*, the photos of printouts taken by a camera). We train the model using a set of paired training examples $\{x_i\}_1^N$ and $\{x_i^V\}_1^N$ where $x_i \in D$ and $x_i^V \in P$. We denote the data distribution as $x \sim pdata(x)$ and $x^V \sim pdata(x^V)$ for brevity. In addition to the mapping function (i.e., the generator), cGans has another component, discriminator C, which aims to discriminate x^V and $p(x)$. We train the cGan models via the following objective function:

$$\mathcal{L}_{cGANs} = \mathbb{E}_{x^V}\ log(C_P(x^V)) + \mathbb{E}_x\ log(1 - C_P(p(x))), \tag{3.2}$$

where the generator p tries to generate images $p(x)$ that look similar to images x^V from the physical domain P, while the discriminator C_P aims to distinguish between *simulated* physical image $p(x)$ and real samples x^V. For D2P, we consider two types of cGANs (equation 3.2) to improve the performance. First, pix2pix model [95] mixed it with a pixelwise reconstruction loss such as $L1$ or $L2$ distance. Second, cycleGAN [222] mixed it with cycle

[1] Other image-to-image translation models can be applied here as well [36, 205].

Table 3.1: Similarity (SSIM) and Dis-similarity (MSE) in comparison with the original digital image, after different several times of digital-to-physical transformation.

Similarity Metric	# of Transformations				
	0	1	2	3	4
SSIM	1.00	0.69	0.54	0.49	0.42
MSE	0.00	1788.75	3180.50	4625.07	4852.89

loss and learned another set of generator ($p^{'} : P \rightarrow D$) and discriminator ($C^{'}_{P}$). The generator $p^{'}$ transforms the *simulated* physical image $p(\mathbf{X})$ back to digital domain and make $p^{'}(p(\mathbf{X}))$ look similar to its original input $\mathbf{X}$; Note that unlike pix2pix, cycleGAN does not require "paired" images for training, which can tolerate the potential misalignment between the digital and physical image. We adopt the network architecture in [95, 222] and follow the training procedure for training our D2P transformation network.

3.3.2 Step 2: Apply Exception over Transformation (EOT)

After training the cGAN model, given an input digital image $\mathbf{X}$, we map the image $\mathbf{X}$ to the *simulated* physical image $p(\mathbf{X})$. We then use $p(\mathbf{X})$ as the "base" and apply the Exception over Transformation (EOT) method to generate adversarial noises. By sampling the geometric transformation of $p(\mathbf{X})$, the EOT method can further improve the robustness of the produced adversarial noise over different viewpoints. Note that our method is operated via *digital simulations*, which incur a low cost. Later, we show that the cGANs can be trained with a one-shot effort using a small set of images (*e.g.*, 200). Once it is trained, the model generalizes well to various different types of images (scalable).

3.3.3 Step 3: Add Noise to Simulated Physical Image

The adversarial noise is then added to the *simulated physical image* $p(\mathbf{X})$ to generate the adversarial image. This is very different from existing works which add noise to the digital image $\mathbf{X}$ [22, 35, 62]. Our design is motivated by an observation from our experiments: after going through physical devices (printers, cameras), the digital images would lose certain features and details due to quantization. Such physical transformation effect is the strongest for the first time and then becomes much weaker when going through multiple rounds of transformations.

Table 3.1 validates this observation. We randomly select 30 images from the ImageNet validation dataset [168]. For each image, we print it out using a printer and retake the photo of the printout using a camera at the frontal view. We consider as one round of digital-to-physical transformation. We then perform multiple rounds of transformation and measure the image similarity (or dis-similarity) to the original clean image. As shown in Table 3.1, we use the Structural Similarity Index (SSIM) [206] and Mean Squared Error (MSE) as the

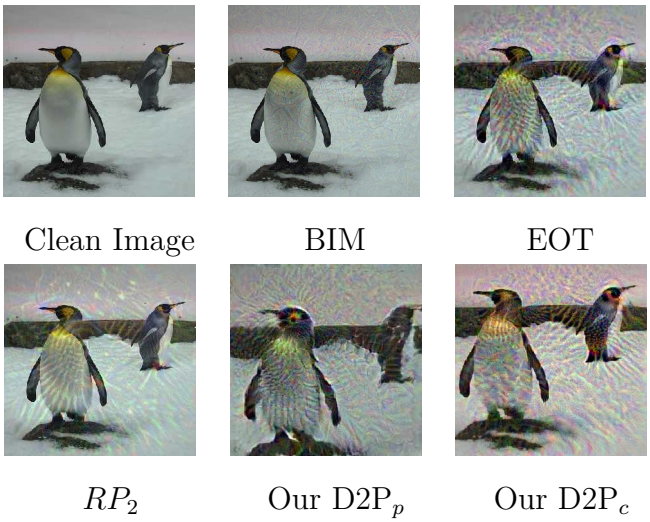


Figure 3.3: Adversarial examples in the physical domain. The targeted attack aims to make the classifier mis-classify the input image from the original label "*king penguin*" to the target label "*kite*".

similarity metric. Our results validate that the loss is more significant during the first round, and then becomes much smaller for the third and fourth round. The result suggests that if we use a (simulated) physical image as the base, the resulting adversarial example is more likely to survive another round of quantization during the attack.

3.4 Experimental Evaluation

We evaluate the effectiveness of adversarial examples in the physical domain with two goals. First, we seek to compare our method with the state-of-the-art over *a much larger-scale* physical domain measurements. Over different experiment settings, we printed and shot over 3000 physical images for a comprehensive evaluation. Second, we seek to examine the *transferability* of our method, *i.e.*, how well an adversarial example optimized for a specific DNN classifier and a pretrained pix2pix/cycleGAN model can transfer to other classifiers, cameras, and printers.

3.4.1 Experiment Setups

We compare our D2P method with existing algorithms including the baseline BIM and the more advanced EOT and RP_2 methods. We choose the widely used Inception-V3 [185] as the *target classifier*, which is pre-trained from the ImageNet dataset [168]. Note that the "physical experiments" require us manually printing images and taking photos, which cannot be fully automated to reach a large scale. To this end, we randomly sampled 102 images (96

Table 3.2: Similarity between the real physical image and the simulated physical image using pix2pix.

Training Size	50	100	150	200	250	300
SSIM	0.37	0.40	0.44	0.45	0.45	0.46
Perception Loss	0.44	0.41	0.40	0.38	0.38	0.37

Table 3.3: Classification confidence and accuracy of adversarial examples. BIM 1's noise level ($\epsilon = 30, \alpha = 0.5$) is the same with all other methods. BIM 2 uses bigger noises ($\epsilon = 70, \alpha = 0.5$).

Method	Digital Domain						Physical Domain					
	Original			Adversarial			Original			Adversarial		
	P(Orig.)	Top1	Top5	P(Adv.)	Top1	Top5	P(Orig.)	Top1	Top5	P(Adv.)	Top1	Top5
Clean	0.94	1	1	0.00	0.00	0.00	0.83	0.97	1	0.00	0.00	0.00
BIM 1	0.00	0.00	0.00	0.95	1.00	1.00	0.56	0.69	0.90	0.00	0.00	0.01
BIM 2	0.00	0.00	0.00	1.00	1.00	1.00	0.29	0.45	0.64	0.00	0.01	0.01
EOT	0.00	0.00	0.00	0.97	1.00	1.00	0.03	0.03	0.14	0.59	0.78	0.90
RP_2	0.00	0.00	0.00	0.97	1.00	1.00	0.10	0.17	0.32	0.40	0.55	0.75
D2P_p	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	0.76	0.91	0.98
D2P_c	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.03	0.75	0.85	0.96
$\text{D2P}_{physical}$	0.00	0.00	0.00	1.00	1.00	1.00	0.01	0.02	0.02	0.71	0.84	0.95

classes) from the ImageNet's validation set (50000 images) [168] as our *Exp Dataset*.

For our D2P method, we trained two types of cGANs to model the digital-to-physical transformation: one is pix2pix (referred as D2P_p), and the other one is cycleGAN (referred as D2P_c). We use 200 training images randomly selected from ImageNet's validation set. To build the ground-truth, we print each image and then re-take the photo to obtain its physical version (Canon PIXMA TS9020 printer and iPhone 6s camera), and use this dataset with paired digital and physical images to facilitate the training. These 200 images *have no overlap* with the 102 images in the *Exp Dataset*. In this way, we can test whether cGANs is indeed generalizable to unseen images. For applying the EOT method, we follow a standard configuration, and consider *scaling* (from 0.5 to 2.0), *rotation* (from $-45°$ to $45°$) and *translation* (from -0.2 to 0.2). The parameters are uniformly sampled.

We only use 200 images for training because our preliminary experiment shows a small training dataset is sufficient. For brevity, we use pix2pix model to demonstrate the impact of training data size (results are similar for cycleGAN). Table 3.2 shows how the size of training dataset affects the quality of the pix2pix output. More specifically, we measure the similarity between the actual physical images and the simulated physical images produced by the pix2pix model based on SSIM and Perception Loss [165]. The similarity scores hit diminishing returns after 200 images. Even though training is a one-shot effort, it is desirable to reduce the manual efforts to produce training data.

We perform *targeted attacks* for all cases. For a given input image, we use the proposed attack method to generate adversarial noises aiming to misclassify the image as the *least-*

likely label (a more difficult attack). For example, suppose an input has a true label of "dung beetle", the "most-likely" label is the label that has the *second highest* classification probability, which is "ground beetle". The "least-likely" label is the one with the lowest probability: "American lobster". Clearly, it is more challenging to cause a mis-classification to the least-likely label. For all the methods, if not otherwise stated, we set the step size $\alpha = 0.5$ and noise level $\epsilon = 30$ to maintain the same level of adversarial noises. As shown in Figure 3.3, the adversarial examples are still visually recognizable as the original label ("King penguin"), but will be misclassified to the target label ("Kite").

Transformation Function for the EOT Method. Under the EOT framework, we use a series of transformations t to generate synthetic images to simulate the effect of scaling, rotation, perspective changes. We use *affine transformation* to construct t. The parameters of the transformation are sampled uniformly at random from the distributions shown in Table 3.4. More specifically, when we optimize the adversarial noises, during each iteration, we generate 10 synthetic images for the noise optimization. Each synthetic image is generated by performing scaling, rotation, and translation on the original image. We randomly select a tuple of 3 parameters (p_s, p_r, p_t) from the respective ranges in Table 3.4 for the transformation. For example, if the sampled parameters are $(0.7, 15°, 0.1)$, we will scale down the size of the image by 70% and then rotate the image by 15 degrees, and then apply translation of 0.1 (to change the perspective) to generate the synthetic image.

Transformation	Minimum	Maximum
Scale	0.5	2.0
Rotation	-45°	45°
Translation	-0.2	0.2

Table 3.4: Distribution of transformations t, where each parameter is sampled uniformly from the specified range.

3.4.2 Experiment Process

Given a digital image $\mathbf{X}$, the experiment process is the following. First, We use the proposed D2P model to generate a simulated physical image $p(\mathbf{X})$ as a base image. Second, we add the adversarial noise to this base image. Third, we print the new image out on a paper as a printout. Fourth, we take a photo of the printout using a phone. Fifth, we send this photo to a DNN classifier, and evaluate the attack performance.

For our baseline methods (BIM, EOT, RP_2), we follow the same process except for the first step. Instead of using the simulated physical image $p(\mathbf{X})$ as a base, we directly use the clean image $\mathbf{X}$ as their base image.

As shown in Figure 3.4, we host a printed image on an L-shaped shelf fixed on a rotational table equipped with a remote controller. This allows us to accurately control the angle of

Figure 3.4: Exp setups.

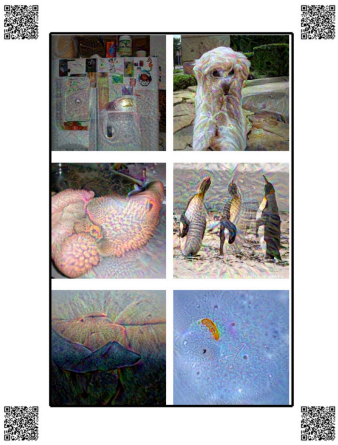


Figure 3.5: Img Printout.

the rotation. The center of the camera is aligned with the center of the printout. To increase efficiency, we take 6 images per printout for the front view (Figure 3.5). Following [113], we place a QR code on the printout so that we can automatically identify, align, and crop the photos. Note that the 6-image setting is only applied for the "frontal-view" experiments. Whenever we take photos from different angles, we print one image at a time as shown in Figure 3.4 to ensure the viewing angle is measured accurately. All photos are taken under the normal indoor lighting. By default, we use a Canon PIXMA TS9020 printer and the iPhone 6s camera. Later, we will examine the transferability using a different printer and camera.

3.4.3 Exp A: Effectiveness of Adversarial Examples

We start with the "frontal view" and examine how likely the adversarial examples can fool the classifier. Table 3.3 shows three key evaluation metrics. First, we report the probability (*i.e.*, confidence) produced by the classifier which indicates the likelihood of the input to be classified as each label. We show the average confidence of the original label (P(Orig.)) and that of the target label (P(Adv.)). Second, after ranking the labels based on the confidence, we show the percentage of images whose original label is ranked top-1 and top-5. Third, we also show how likely the target label is ranked at the top-1 and top-5. In Table 3.3, the "'*clean*" row refers to clean images without attacks. The classifier has a perfect classification accuracy (100%) in the digital domain and a near-perfect performance in the physical domain. A successful adversarial example will suppress the original label (low P(Orig.) — low top-1 and top-5 ratio for the original label), and promote the target label (high P(Adv.) — high top-1 and top-5 ratio for the target label).

We have *four key observations* from the attack results. First, as shown in the left half of the table, the digital versions of the adversarial images are highly successful. Across all the methods, 100% of the original labels are dropped out of top-5, and the target label is always classified as the top-1. This shows that in the digital domain, a classifier can be extremely

vulnerable to adversarial attacks.

Second, as shown in the right half of the table, adversarial examples are more difficult to succeed in the physical domain. The top-1 accuracy of the target label dropped significantly for BIM to 0.00 and 0.01. The results suggest that the basic methods do not work in the physical domain. Advanced methods such as EOT and RP_2 have a better performance, which confirms the advantage of optimizing over simulated geometric transformations.

Third, both of our D2P models outperform existing methods by a large margin. Compared to EOT and RP_2, our method significantly improves the target label's ranking. For example, using D2P_p, the top-1 accuracy of the target label is improved to 0.91 from 0.55 and 0.78. The top-5 accuracy of the target label is improved to 0.98. In addition, our method successfully reduces the original label's top-1 and top-5 accuracy to 0. These results demonstrate the benefits of using a *simulated physical image* as the base to generate adversarial examples and the cGANs have successfully captured the patterns of D2P transformation.

Fourth, D2P_p slightly outperforms D2P_c in the attacking results. D2P_c uses the cycleGAN for learning the digital-to-physical transformation. The simulated physical images are more authentic compared with the real physical images because the training does not suffer from potential misalignment between the digital and physical images. As evidence, we measure the average Perception Loss, a metric to assess the visual dissimilarity [165] between the actual physical image and the simulated one. We find that cycleGAN indeed has a lower loss (0.28) than the pix2pix model (0.38). Although cycleGAN makes the generated image more faithful to the real physical image (see the example in Figure 3.3), it also preserves more features of the original image which makes the attack more difficult. The attacking performance of D2P_c is slightly weaker than that of D2P_p.

Given the good performance of D2P, a natural question is whether the performance would be even better if we directly use the physical image as the base ($\text{D2P}_{physical}$). This represents the best base image that the D2P model can output. As shown in Table 3.3, the result is counter-intuitive, as D2P_c and D2P_p perform slightly better than $\text{D2P}_{physical}$. A possible explanation is that the performance gain may come from the feature loss during the quantization. The simulated physical images produced from the cGAN model exhibit slight distortions compared to the corresponding physical images. The feature loss makes the simulated images slightly easier to attack. In the rest of the paper, we use D2P_p to examine the robustness of adversarial examples.

3.4.4 Exp B: Robustness against Viewing Angles

Next, we examine the *robustness* of the adversarial examples by changing the viewing angles. The goal is to assess a realistic scenario where the target system (*e.g.*, self-driving car) may take photos from different angles to classify an object. A robust adversarial example should remain effective under different viewing angles. In this experiment, we test 9 different angles ranging from -60° to 60° by rotating the turntable with a 15-degree increment at a time to

Table 3.5: Transferability of adversarial examples. "Base" represents the result of the original configuration of Exp. A and B. We then examine the performance of adversarial examples under different phone cameras, printers, and classifiers.

Model	Our Method D2P$_p$						EOT					
	Original			Adversarial			Original			Adversarial		
	P(Orig.)	Top1	Top5	P(Adv.)	Top1	Top5	P(Orig.)	Top1	Top5	P(Adv.)	Top1	Top5
Base	0.00	0.00	0.00	0.76	0.91	0.98	0.03	0.03	0.14	0.59	0.78	0.90
Diff. phone	0.00	0.00	0.01	0.80	0.90	0.97	0.03	0.10	0.14	0.49	0.68	0.83
Diff. printer	0.00	0.00	0.00	0.87	0.97	1.00	0.02	0.03	0.05	0.80	0.91	0.99
Xception	0.00	0.00	0.01	0.37	0.54	0.79	0.06	0.11	0.24	0.22	0.45	0.63
ResNet	0.01	0.01	0.01	0.24	0.35	0.57	0.05	0.06	0.19	0.15	0.17	0.38
MobileNet	0.00	0.00	0.02	0.23	0.37	0.56	0.05	0.07	0.21	0.14	0.21	0.49

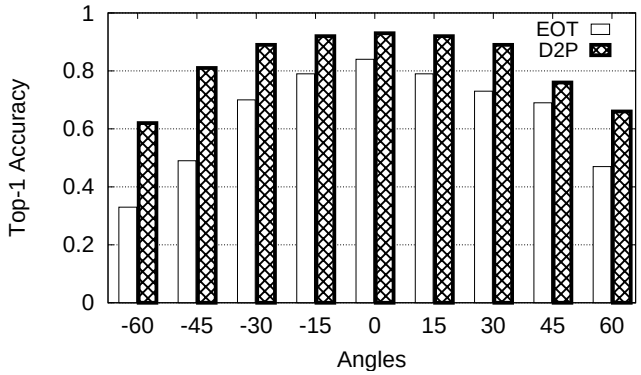


Figure 3.6: Top-1 accuracy of target label for the adversarial examples under different viewing angles.

take photos. To accurately capture the angle, we print one image at a time (instead of 6 images per paper). For this experiment, we only compare our D2P$_p$ method with the best performing baseline, the EOT method ($\epsilon = 30$).

Figure 3.6 shows that both methods perform reasonably well under different angles. This is largely benefited from the synthetic *geometric transformations* used by both methods. Our D2P method has a better performance compared to EOT, and the advantage is more significant at larger angles. For example, at the frontal view, our top-1 accuracy is 0.91 and EOT's is 0.78. When the image is turned by 45 degrees, our method still has a top-1 accuracy of 0.62 while the accuracy of EOT degrades to 0.33. The results confirm the robustness of our adversarial examples. Recall that our digital-to-physical model was trained only using the *front view* images. The result shows that the transformation helps to generalize better the attack effectiveness (compared to EOT) to other previously *unseen* situations (*i.e.*, images captured from different view angles).

	Before retraining		After retraining	
	Clean	Adv examples	Clean	Adv examples
Accuracy of original class	100%	0%	97%	24%
Accuracy of adversarial class	0%	91%	0%	23%

Table 3.6: Statistics about before adversarial retraining and after adversarial retraining.

3.4.5 Exp C: Transferability of Adversarial Examples

We validate the transferability of the proposed adversarial examples. So far, we were using a specific DNN model (Inception-V3), camera (iPhone 6S), and printer (Canon PIXMA TS9020) to generate adversarial examples. Below, we examine how robust these adversarial examples are when we (1) print the adversarial images with a different printer; (2) take photos with a different camera, and (3) classify the physical images with a different classifier. This simulates a practical scenario where the attacker does not have full knowledge of the target system. all adversarial examples are generated in the same setting as before (Inception V3). Next, we test the images by changing one condition at a time. As shown in Table 3.5, we first change the iPhone to an Android Phone (Motorola Moto G5 Plus). Then we test a different printer (Xerox Phaser 7500). Finally, we change the DNN architecture to Xception [41], ResNet [79] and MobileNet [88].

Table 3.5 shows that using a different printer or camera does not significantly affect the results. With an Android phone, the top-1 accuracy for the target label is still as high as 0.9. When we use a different printer (Xerox), the result actually gets better (top-1 accuracy is 0.97). We believe that this is because the Xerox printer is a laser printer with a higher DPI (1600). The Canon printer used to train the pix2pix network is an ink printer with 600 DPI. Therefore, the quantization effect has been over-estimated during training, and the performance improves when the adversarial examples are printed out by a high-DPI printer.

The DNN architecture, however, does have an impact. We observe that Xception performs better than ResNet and MobileNet, which is likely due to the fact that Xception uses the same image size (299×299) as the original Inception-V3, while the other two would reshape the images before classification. We suspect the digital-to-physical transformation also plays a role. To validate this hypothesis, we performed an experiment where we directly feed the *digital version* of the adversarial images into the target classifiers. We observe the performance degradation is much smaller on digital images. Consistently across all settings, we show that our method has a better transferability compared with EOT. This indicates that our cGANs model has captured generalizable characteristics of the physical domain transformation.

3.4.6 Exp D: Retraining using D2P

Finally, we show how **D2P** improves the robustness of machine learning models by adversarial retraining [167]. We use adversarial examples generated by **D2P** to retrain the inception model. And these examples are in the digital domain, in other words, we do not use physical images of adversarial examples generated by **D2P** to do adversarial retraining. And we test the model using physical images of adversarial examples generated by **D2P**. Table 3.6 shows the difference between using retraining and without using retraining. The robustness of the model is improved, the accuracy of the adversarial class is dropped to 23% from 91%. Although we can not cover all of them to the original class, further investigation shows the average confidence of the adversarial class is only 0.11. And we note that the confidence before retraining is 0.76. It indicated as the defender, we have the option to reject these adversarial examples. For example, for self-driving car systems, we can remind the driver not to trust these images because we don't have enough confidence. Overall, this indicates **D2P** could improve the robustness of machine learning models.

3.5 Discussion and Conclusion

In this chapter, we explore the feasibility of generating robust adversarial examples that can survive in the physical world. We propose the **D2P** method to simulate the complex effect introduced by physical devices to construct more robust adversarial examples. Our results show that the simulated transformation helps improve the attack effectiveness to other *unseen* or *uncontrolled* situations such as different viewing angles, printers, and cameras. Using **D2P**, we can generate more *realistic* adversarial examples to assist the troubleshooting of under-trained regions and augment the training data for model retraining [167] or adversary detection [212]. By adding our adversarial examples into the training data, the re-trained classifier become more robust against attacks.

Extending D2P. One advantage of **D2P** is it is easy to combine **D2P** with other attacks. **D2P** contains two major steps. The first step is to train Image-to-Image translation networks (pix2pix), which is a one-time effort. The second step is to apply EOT on the simulated image (the output from the first step). One way to extend **D2P** with other attacks is to replace EOT in the second step, in other words, apply other attacks on the simulated image. It could improve the robustness of other attacks in the physical domain because of less quantization loss and feature loss as shown in Section 3.3.

Another extension of **D2P** is about defense. It is natural to ask can we use **D2P** to defend against other new types of attacks. As shown in section 3.4.6, **D2P** could help the models defend against physical attacks generated by using **D2P**, however, they are effective because training and testing examples are from the similar distribution. Therefore, if the new types of the attacks are very different from **D2P**, it is likely the retrained model could not defend

against other attacks. And this is the limitation of model retraining.

3D Adversarial Examples. EOT [22] further extend their works to produce 3D adversarial examples by modeling the 3D rendering as a transformation under EOT. Given a textured 3D object, it optimizes the texture such that the rendering is adversarial from any viewpoint. It also considers a distribution that incorporates different camera distances, lighting conditions, translation and rotation of the object, and solid background colors. In the experiment, it uses a 3D-printer to print out a 3D-printed adversarial turtle, and it is consistently classified as a rifle.

Chapter 4

Bot Detection: Data Augmentation to Tackle Data Scarcity Problem

4.1 Introduction

In this chapter, we seek to robustify the machine learning models for detecting the attackers when we only the limited data of attackers. Although machine learning has shown great success for building defenses in many security applications [20, 23, 45, 47, 51, 138, 178, 181, 193], these ML based applications often require "labeled data" to train a good detection model [20, 193]. And it is highly expensive to obtain a large amount of labels (*e.g.*, via manual efforts).

Our key methodology to address this concern is to perform *data synthesis*. 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* attackers' samples. The assumption is that the benign samples are relatively more stable and representative than the limited attackers data. We thus synthesize new attackers samples for the unoccupied regions in the feature space by differentiating "clustered regions" and "outlier regions". At the clustered regions (which represent common user/attackers 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 attackers variants), our data synthesis is more aggressive to fill in the space. Based on these intuitions, we designed a customized Generative Adversarial Network (GAN) with two complementary generators to synthesize clustered and outlier data simultaneously.

We 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.

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 lifecycle 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.

4.2 Background and Goals

4.2.1 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) [52, 122, 169], most bots are operated to engage malicious actions such as spam, scam, click fraud and data scrapping [38, 50, 51, 53, 76, 190, 191, 201]. 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" [199]. 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 pre-defined rules [176]. 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 [49, 111, 160]. A common way is supervised training with labeled bot data and benign user data [53, 68, 99, 190]. There are also unsupervised methods [15, 94, 130], but they are often limited in accuracy compared to supervised methods.

4.2.2 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) [53, 68, 190, 201]. 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

Table 4.1: Dataset summary.

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

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.

4.3 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. 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.

4.3.1 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 4.1 shows the summary of the datasets. For anonymity purposes, we use A, B, C to represent the 3 websites. For all three websites, we obtained their logs in August 2018 (08/01

to 08/31). Then we collected data in January 2019 (01/08 to 01/31) and September 2019 (09/01 to 09/30) for two websites (A, and B). We were unable to obtain data from website C for the January and September of 2019 due to its service termination with 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.

4.3.2 Reprocessing: IP-Sequence

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

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

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

4.3.3 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 exami-

Table 4.2: Estimated false positives of rules on IP-sequences.

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

nation 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 [14, 27, 140, 213]. However, even the most advanced attack [213] 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 [173]. 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 [143]. 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 4.2. For example, for website A, the rules matched 42,487 IP-sequences. Among them, 38,294 sequences have received a CAPTCHA, and only in 4 out of 38,294 (0.01%) users solved the CAPTCHA. This confirms the extremely high precision of rules. As a trade-off, the rules missed many

Table 4.3: 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	-	-	-	-	-	-

real bots, which are further discussed in Section 4.4.1.

4.3.4 Problem Definition

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

Scope and Limitations. Certain types of attackers are out of scope. Attackers that hire human users to solve CAPTCHAs [143] 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).

4.4 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 4.5).

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

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

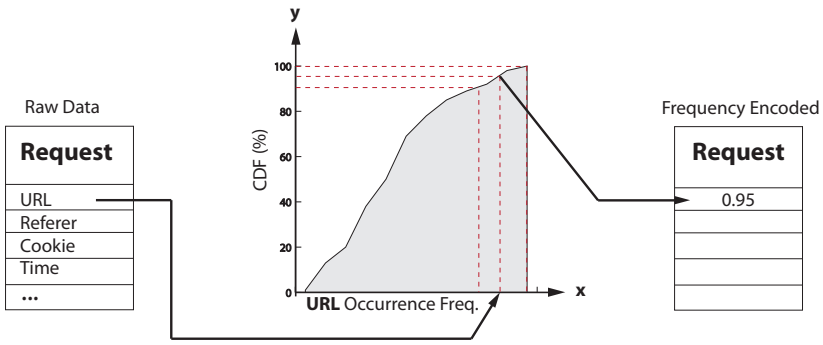


Figure 4.1: Example of frequency encoding for the visited URL.

4.4.1 Phase I: Filtering Simple Bots

As discussed in Section 4.3.3, 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 4.6). This requires Phase II to detect the advanced bots that bypassed the rules. Table 4.3 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).

4.4.2 Phase II: Machine Learning Model

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

[1]We set $T = 30$ because the sequence length T needs to be reasonably large to obtain meaningful patterns [187]. 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 [148, 155, 180] which are complementary to our goals.

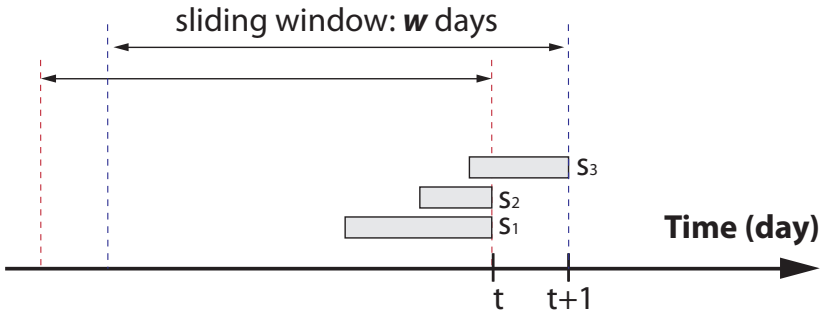


Figure 4.2: Example of sliding window for feature encoding. S_1 and S_2 are IP sequences formed on day t, and S_3 is formed on day $t+1$. The feature vendors are encoded using the past w days of data.

Table 4.4: 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

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 [137]. Word2Vec can be applied to process network traffic [204]: URLs that commonly appear at the same position of a sequence will be embedded to vendors with a smaller distance. Embedding methods are useful for *offline data processing*, and is not suitable for a real-time system. Word2Vec requires using a large and *relatively stable* dataset to generate a high quality embedding [162], 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 4.1, given a request, we encode the URL hash based on its occurrence frequency in the past. This example URL appears very frequently in the past at a 95-percentile. As such, we map the URL to an index value "0.95". In this way, URLs that share a similar occurrence frequency will be encoded to a similar index number. This scheme can easily handle new/rare instances: any previously-unseen entities would be assigned to a low distribution percentile. We also don't need to manually divide the buckets but can rely on the data distribution to generate the encoding automatically. The feature value is already normalized between 0 and 1.

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

Table 4.4 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 [204]. 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 [85, 136]. 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 [37], 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).

Table 4.5: The detection results of LSTM model on "advanced bots".

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

Table 4.6: 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

4.4.3 Evaluating The Performance

We evaluate our model using data from August 2018 (advanced bots). We followed the recent guideline for evaluating security-related ML models [156] to ensure result validity.

Training-Testing Temporal Split. We first ensure the *temporary training constraint* [156], 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 4.5.
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

Table 4.7: 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

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

As shown in Table 4.5, the precision and recall of the LSTM model are reasonable, but not extremely high. For example, for website B, the precision is 0.888 and the recall is 0.877. The worst performance appears on C where the precision is 0.789 and the recall is 0.730. Since our model is focused on advanced bots that already bypassed the rules, it makes sense that they are difficult to detect.

Table 4.6 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.

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

We run a quick experiment with limited training data (Table 4.7). 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).

4.5 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) [33, 158]. In our case, the main problem is the training data is too sparse to train an accurate model in the first place. We design a data synthesis method called ODDS. The key novelty is that our data synthesis is distribution-aware — we use different generalization functions based on the characteristics of "outliers" and "clustered data" in the labeled data samples.

4.5.1 Motivation of ODDS

Training with limited data tends to succumb to overfitting, leading to a poor generalizability of the trained model. Regularization techniques such as dropout and batch normalization can help, but they cannot capture the data invariance in unobserved (future) data distributions. A promising approach is to synthesize new data for training augmentation. Generative adversarial network (GAN) [71] 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 [215], 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

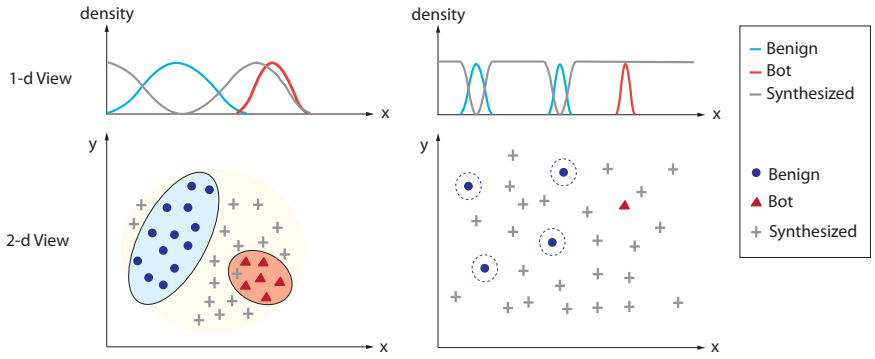


Figure 4.3: Illustrating data synthesis in the clustered data region (left) and the outlier data region (right).

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 4.3 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.

4.5.2 Overview of the Design of ODDS

At a high level, ODDS contains three main steps. *Step 1* is a prepossessing step to learn a latent representation of the input data. We use an LSTM-autoencoder to convert the input feature vectors into a more compressed feature space. *Step 2*: we apply DBSCAN [61] 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, $\mathbf{M} = \{\mathbf{X}_1, \ldots, \mathbf{X}_N\}$ is a labeled training dataset where $\mathbf{X_i}$ is an IP-sequence. $\mathbf{X_i} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ where $\mathbf{x}_t \in \mathbb{R}^d$ denotes the original feature vector of the t_{th} request in the IP-sequence.

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

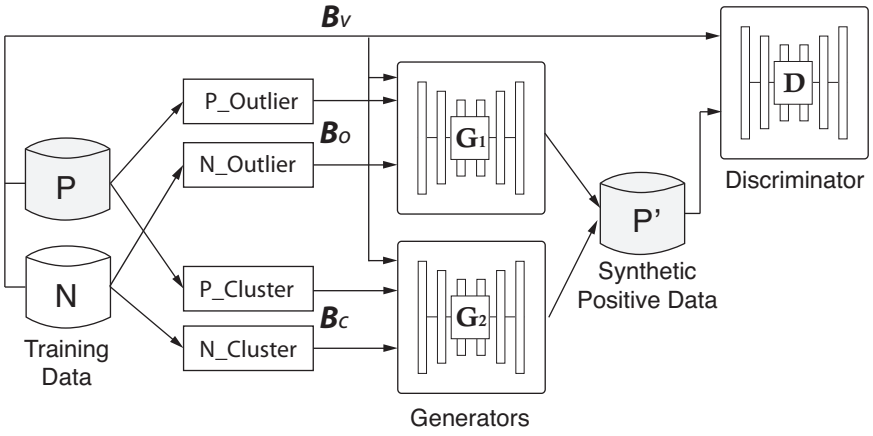


Figure 4.4: The data flow of ODDS model. "Positive (P)" represents bot data; "Negative (N)" represents benign data.

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

Data Clustering. In the second step, we use DBSCAN [61] 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 [171]. 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 $\mathcal{B}_o$.

4.5.3 Formulation of ODDS

As shown in Figure 4.4, 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 4.3 (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:

Table 4.8: Training with 100% or 1% of the training data (the first two weeks of August-18); Testing on the last two weeks of August-18.

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

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

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.

Implementation. To optimize the objective function of generators, we adapt several approximations. To minimize $\mathcal{H}(p_G)$, we adopt the pull-away term [48, 217]. To estimate p_m and p_b, we adopt the approach proposed by [146] which uses a neural network classifier to approximate.

4.6 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?

4.6.1 Experiment Setup

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

Hyperparameters. Our basic LSTM model (see Section 5.3.3) 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 5.3.3, and a non-deep learning model Random Forest [30]. We also include an anomaly detection method as a baseline. We select a GAN-based method called OCAN [219] 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.

4.6.2 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 4.5 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 4.8 (the upper half), presents a more detailed break up of performance

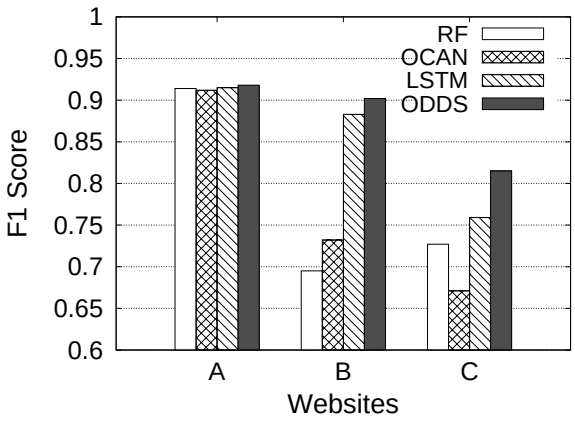


Figure 4.5: Training with 100% training data in August 2018.

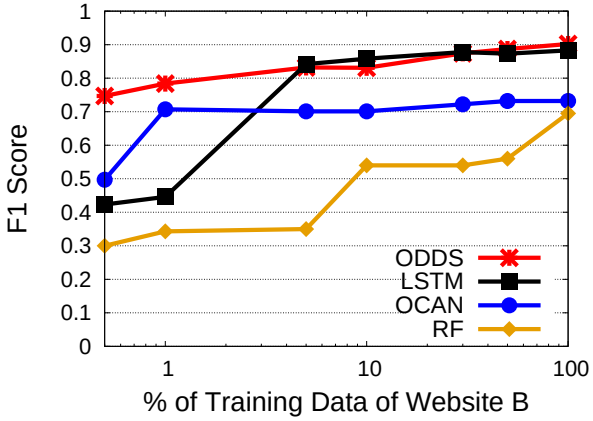
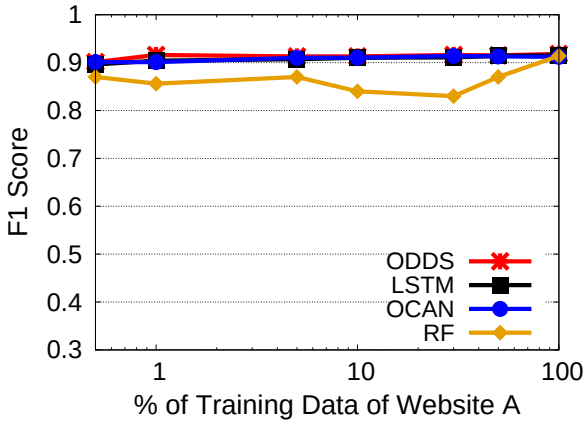
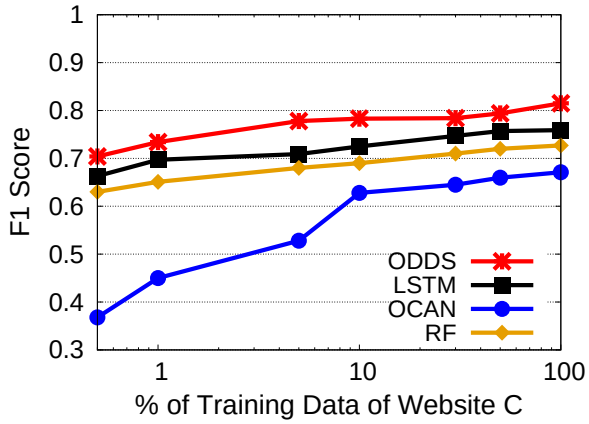


Figure 4.6: Training with x% of training data in August 2018 (B).



(a) Website A



(b) Website C

Figure 4.7: Training with x% of training data in August-18 (A and C).

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.

Table 4.9: 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

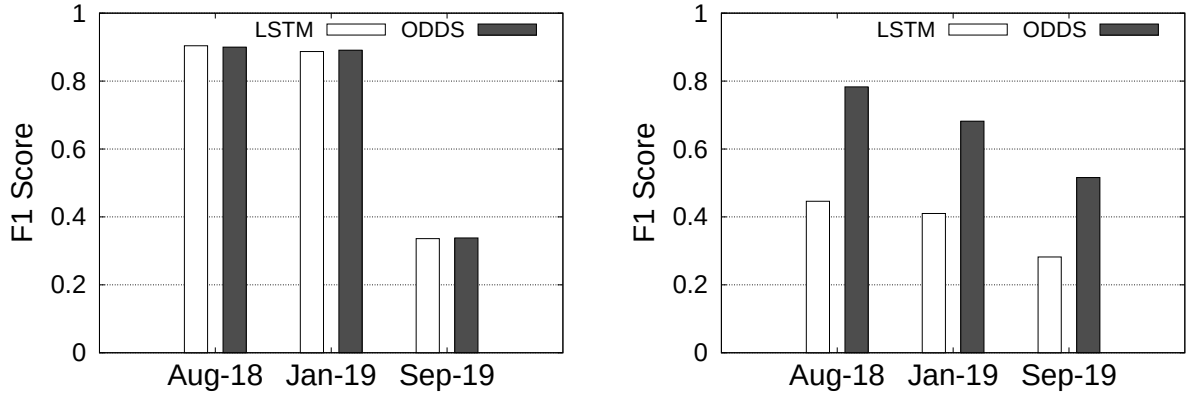
4.6.3 Training with Limited Data

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

As briefly shown in Section 4.4.3, the performance of the LSTM model degrades a lot when training with 1% of the data, especially for website B. Here, we repeat the same experiment and compare the performance of ODDS and LSTM.

Figure 4.6 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 4.8, the performance gain is mostly coming from "recall", indicating the synthesized data is helpful to detect bots in the unknown distribution.

Figure 4.7 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 4.9. First, we compute the average Euclidean distance between the bot and benign data points in the August training set (averaged across all bot-benign pairs). A larger average distance indicates that bots and benign users are further apart in the feature space. The result shows that A clearly has a larger distance than that of B and C. This confirms that bots in A are already highly different from benign users, and thus it is easier to capture behavioral differences using just a small sample of the data. We also calculate the average distance between the bots in the training set and the bots in the testing set. A higher distance means that the bots in testing data have behaved more differently from those in the training data (and thus are harder to detect). We find A has the lowest distance, suggesting bot behaviors remain relatively consistent. B has the highest distance, which requires the



(a) Website A

(b) Website B

Figure 4.8: 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.

detection model to generalize well in order to capture the new bot behaviors.

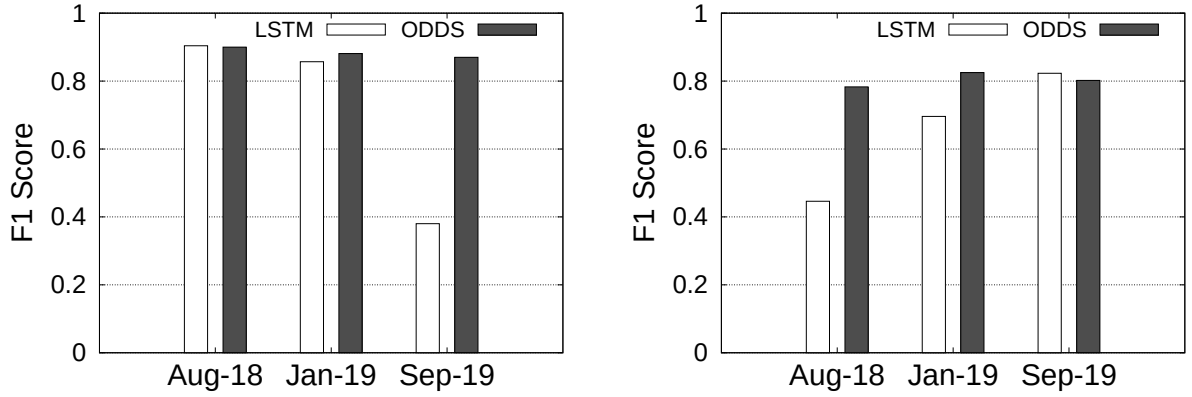
4.6.4 Generalizability in the Long Term

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 4.8. 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



(a) Website A

(b) Website B

Figure 4.9: 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 4.10: 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

is initially trained with only 1% of August-2018 training dataset (first two weeks) and tested in the last two weeks of August 2018. Once it comes to January 2019, we add the first 1% of the January 2019 data to the original 1% of August 2018 training data to re-train the model. This forms a January model, which is then tested on the rest of the data in January. Similarly, in September 2019, we add the first 1% of the data in September to the training dataset to retrain the model. In practice, one can choose to gradually remove/expire older training data for each retraining. We did not simulate data expiration in our experiments since we only have three months of data.

As shown in Figure 4.9 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.

Table 4.11: 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 4.12: 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%

4.6.5 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 4.10, 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.

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

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

Data synthesis has its limitations. To answer this question, we analyze the errors produced by ODDS, including false positives (FP) and false negatives (FN). In Table 4.11, 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 4.12. 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 4.12, we also analyzed the FPs of ODDS. We find that FPs are closer to the malicious region (0.402) than to the benign region (0.644), which explains why they are misclassified as "bots". Note that both 0.644 and 0.402 are high distance values, confirming that FPs are outliers far away from other data points. When we check their 100 nearest neighbors, we surprisingly find that 82% of their nearest neighbors are benign. However, a closer examination shows that most of these benign neighbors turn out to be *other FPs*. If we exclude other FPs, only 9% of their 100 nearest neighbors are benign. This confirms that FPs misclassified by ODDS behave differently from the rest of the benign users.

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

4.6.7 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 [33] 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

Table 4.13: 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

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 [151], 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 4.13 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 [33] 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.

4.7 Discussion

Adversarial Poisoning. 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.

Anomaly Detection. Our method and other anomaly detection methods share a similar assumption that the benign data is relatively more stable. In our dissertation, we selected OCAN, and another popular anomaly detection method called One-class SVM [129]. One-class SVM aims to separate one class of samples from all the others by constructing a hyperplane around the data samples. In this experiment, we use "benign" data as the known class. We choose a threshold t based on the better validation f1 score. As shown in Table 4.14, 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 4.14: 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

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 [56] for model updation.

4.8 Conclusion

In this chapter, 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.

Chapter 5

Phishing Websites Detection: Measurement-Driven Analysis to Understand Adaptive Attackers

5.1 Introduction

In this chapter, we seek to robustify the machine learning models to detect adaptive attackers. In security applications, attackers are constantly evolving to deceive users and evade the detection. These evasion techniques are not clear, and these evasion techniques are likely to render existing detection methods ineffective.

Our main methodology to address this concern is to *measure and build robust features*. We use *phishing websites detection* as our example application. In the high level idea, there are two evasions techniques that are applied to these squatting phishing websites [193]. The first evasion is domain name evasion via domain squatting techniques [87, 106, 141]. The second evasion technique is web content evasion via code obfuscation, string obfuscation. We design a novel measurement system SquatPhi to search and detect squatting phishing domains. We start by detecting a large number of "squatting" domains that are likely to impersonate popular brands. Then, we build a distributed crawler to collect the webpages and screenshots for the squatting domains. Finally, we build a machine learning classifier to identify squatting phishing pages.

For content level evasion, a key novelty is that our classifier is built based on a careful measurement of the evasion techniques used by real-world phishing pages These evasion techniques are likely to render existing detection methods ineffective. To this end, we apply visual analysis and *optical character recognition* (OCR) to extract key visual features from the page screenshots (particularly the regions of the login form). The intuition is that no matter how attackers obfuscate the HTML content, the visual presentation of the page will still need to look legitimate to deceive users.

For domain name evasion, instead of building a machine learning classifier, we seek to understand how robustness for the existing browsers defending against the phishing pages that perform evasion in the domain names, in particular for the homograph IDN domain [86]. To mitigate this risk, browsers have recently introduced defense policies. Commonly, browsers

implement rules to detect homograph IDNs that are likely impersonating other legitimate domain names [73]. Once detected, browsers will no longer display the Unicode, but display their Punycode version. Punycode code is initially designed to translate IDNs to ASCII Compatible Encoding so that they can be recognized by legacy protocols and systems. For example, the Punycode for apple.com with the Cyrillic "a" is "xn—pple-43d.com/". By displaying this Punycode in the address bar, browser vendors try to protect users from deception. However, it is not yet well understood regarding how these policies/rules are constructed and how effective they are.

In order to understand how browsers defend against IDN homograph attacks, we construct more than 9,000 testing cases to examine 1) the browser's enforcement of known IDN policies; and 2) possible ways to bypass existing policies. To run a large number of tests over various browsers and platforms, we build a tool to instrument browsers to load testing IDNs while video-recording browsers' reactions. Based on the recorded videos, we automatically analyze how browsers handle different IDNs.

We run a user study to understand user perception of homograph IDNs. Participants examine a series of website screen-shots The domain names of the webpages are a mixture of real domain names and homograph IDNs (including those that are blocked by Chrome and those that can bypass Chrome). We study users' ability to judge the authenticity of the domain names under mild priming. Our study shows that users are significantly better at identifying real domain names (94.6% success rate) than identifying homograph IDNs. For example, participants only have a success rate of 48.5% on IDNs blocked by Chrome. In addition, we find homograph IDNs blocked by Chrome are indeed more deceptive than those not blocked. Even so, the homograph IDNs that Chrome missed can still deceive users for 45.8% of the time, posing a nontrivial risk. Finally, we show that users' education level, computing background, age, and gender have a significant correlation with their performance in judging domain authenticity, while website popularity and category are not significant factors.

In summary, our key contributions are:

- *First*, we propose a novel end-to-end measurement framework SquatPhi to search and detect squatting phishing pages from a large number of squatting domains.[1]
- *Second*, we systematically test browser-level defenses against homograph IDNs. We show all of the tested browsers have weaknesses in their policies and implementations, making it possible for homograph IDNs to bypass the defense.
- *Third*, we develop a tool to automatically perform black-box testing on browser IDN policies across browser versions and platforms. The tool can be used to monitor and test future browsers.
- *Forth*, we perform a user study to examine user perception of homograph IDNs, and demonstrate the need to enhance the current defense against IDN-based phishing. We have disclosed our findings to related browser vendors.

[1]We open-sourced our tool at https://github.com/SquatPhish.

5.2 Background

We start by briefly introducing the background of phishing pages.

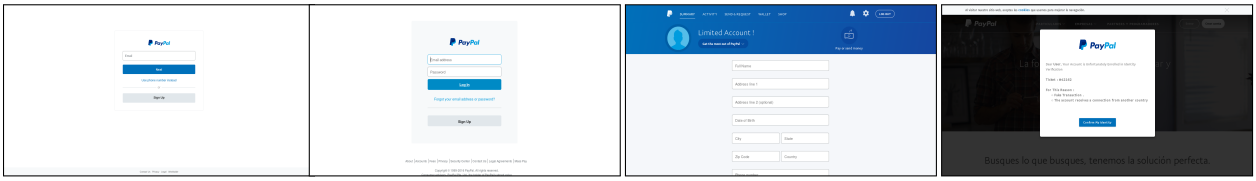
Phishing Web Pages. Phishing has been widely used by cybercriminals to steal user credentials and breach large networks. Typically, attackers would impersonate a trusted entity to gain the victim's trust, luring the victim to reveal important information. Phishing pages often act as the landing pages of malicious URLs distributed by phishing emails [112], SMS [161], or social network messages [59]. The phishing pages usually contain a form to trick users to enter passwords or credit card information.

As phishing attacks become prevalent [4], various phishing detection methods have been proposed, ranging from URL blacklisting [26] to visual similarity based phishing detection [132] and website content-based classification [208]. Visual similarity-based phishing detection [132] aims to compare the original webpages of popular brands to suspicious pages to detect "impersonation". Machine learning based methods [208] rely on features extracted from the HTML source code, JavaScript, and the web URLs to flag phishing websites. As phishing attacks evolve, we are curious about the potential evasion techniques used by attackers in practice.

Domain Name Squatting. Domain name squatting is the act of registering domain names that are likely to cause confusions with existing brands and trademarks. Domain name squatting has led to abusive activities such as impersonating the original websites to steal traffic, and distribute ads and malware. A squatting domain usually shares many overlapping characters at a targeted domain. Common squatting techniques include bit mutation [147], typo spelling [141] and homograph imitating [106]. Internationalized domain names (IDN) can be used for domain squatting domains, since IDNs can have a similar visual representation as the target domains after encoding. Existing works have performed real-world measurements and found homograph IDNs that impersonate popular domain names [39, 115, 121, 183].

Squatting domains can cause trouble to users as well as the target brands. For example, users often mis-type the domain name of the website they want to visit in the address bar (*e.g.*, typing facbook.com for facebook.com). As a result, users could be visiting a website hosted under a squatting domain. Speculators register squatting domains of popular brands and resell them at a much higher price. Sometimes, popular brands (*e.g.*, big banks) have to purchase squatting domains that targeting their websites so that they can redirect users back to the correct websites [6].

Domain Squatting for Phishing. Squatting domains are naturally powerful to conduct phishing attacks since the domain name looks similar to that of a trusted website. We refer phishing pages hosted under squatting domains as *squatting phishing pages*. More formally, a squatting phishing page (P_s) has two properties: (1) it has a squatting-based domain (S); and (2) its webpage contains deceptive phishing content (W). $P_s = S \vee W$.



(a) The original paypal page. (b) Phishing page (distance 7). (c) Phishing page (distance 24). (d) Phishing page (distance 38).

Figure 5.1: An example of page layout obfuscation of phishing pages (paypal).

Internationalized Domain Name (IDN). A domain name is an identification string for Internet hosts or services. Through the Domain Name System (DNS), a user-readable domain name can be mapped to its corresponding IP address. Originally, domain name only allowed ASCII (English) letters, digits and hyphens [139]. In 2003, Internationalized Domain Name (IDN) was introduced to allow people, especially non-English speakers, to use characters from their native languages to create domain names [93]. The new specification supports Unicode characters, which cover more than 143,000 characters from a variety of languages (154 scripts, divided into 308 blocks) [198].

Punycode. The challenge of using IDN is that non-ASCII characters are not supported everywhere. To maintain compatibility with existing protocols and systems, there needs to be a way to convert IDNs to ASCII Compatible Encoding (ACE) strings. The standardized mechanism is called Internationalizing Domain Names in Applications (IDNA) [44, 163]. IDNA converts Unicode labels to an ASCII Compatible Encoding (ACE) label which is also called Punycode. Punycode always starts with "xn—". For example, Unicode string "bücher.de" is mapped to Punycode "xn—bcher-kva.de." IDNA has been adopted by browsers and email clients to support IDNs. Before sending a DNS query for IDN, the domain name is usually translated to its Punycode first to ensure the success of the DNS resolving.

5.3 Detecting Phishing Pages Performing Evasive Page Contents

5.3.1 Collect Ground Truth Phishing Pages

To develop an effective phishing detection system, we need to understand whether and how phishing pages are currently and actively evading common detection methods in practice based on ground-truth phishing pages. In the high level idea, we crawl and manually valid 1,731 ground truth phishing pages from PhishTank [5][2].

[2]We refer the readers to the paper [193] for complete details.

5.3.2 Content Level Evasion Measurement

Based on the ground-truth data, we next examine the common evasive behavior of phishing pages. We will use the measurement results to derive new features to more robust phishing page detection. Our evasion measurement focuses on three main aspects: the image layout, the string text in the source code, and obfuscation indicators in the javascript code. These are common places where adversaries can manipulate the content to hide its malicious features, while still giving the web page a legitimate look. For this analysis, we focus on the web version of the pages. We find that 96% of the pages on PhishTank have the same underlying HTML sources for both the web and mobile versions. This indicates that the most attackers did not show different pages to the web and mobile users (*i.e.* no cloaking).

Layout Obfuscation. Many phishing detection methods assume that the phishing pages will mimic the legitimate pages of the target brands. As a result, their page layout should share a high-level of similarity [132]. Phishing detection tools may apply some fuzzy hashing functions to the page screenshots and match them against the hash of the real pages. To examine the potential evasions against page layout matching, we compute the Image hash [3] to compare the visual similarity of the phishing pages and the real pages of the target brands. The (dis)similarity is measured by the hamming distance between two image hashes.

We find that layout obfuscation is widely applied, and phishing pages often change their layout greatly to evade detection. Figure 5.1 shows a real example in our dataset for brand paypal. The left-most page is the official paypal page. The other 3 pages are phishing pages with different image hash distances 7, 24 and 36 respectively compared to the real pages. With a distance of 7, the phishing page is still visually similar to the original page. When the distance goes to 24 and 36, the pages look different from the original pages but still have a legitimate looking. Those pages would be easily missed by visual similarity based detectors.

String Obfuscation. String obfuscation is hiding important text and keywords in the HTML source code. For example, attackers may want to hide keywords related to the target brand names to avoid text-matching based detection [104]. For example, in a phishing page that impersonates paypal, we find that the brand name string is obfuscated as "PayPaI", where the "l" (the lower case of "L") is changed to "I" (the upper case of "i"). Another common technique is to delete all related text about the brand name paypal but instead put the text into images to display them to users. From the users' perspective, the resulting page will still look similar.

We perform a simple measurement of text string obfuscation by looking for the brand name in the phishing pages' HTML source. Given a phishing page (and its target brand), we first extract all the texts from the HTML source. If the target brand name is not within the texts, then we regard the phishing page as a string obfuscated page. Table 5.1 shows the percentage of string obfuscated pages for each brand. For example, 70.2% of microsoft phishing pages are string obfuscated. 35.3% of facebook phishing pages are string obfuscated. This suggests that simple string matching is less likely to be effective.

Code Obfuscation. Javascript code may also apply obfuscation to hide their real purposes. This is a well-studied area and we use known obfuscation indicators to measure the level of code obfuscation in the phishing pages. Obfuscation indicators are borrowed from FrameHanger [194]. According to previous studies [103, 210], string functions (*e.g.*, fromChar and charCodeAt), dynamic evaluation (*e.g.*, eval) and special characters are heavily used for code obfuscation. For each phishing page, we download and parse the JavaScript code into an AST (abstract syntax tree). We then use AST to extract obfuscation indicators.

Brand	String Obfuscated	Code Obfuscated
Santander	30 (100%)	4 (13.3%)
Microsoft	200 (70.2%)	127 (44.6%)
Adobe	38 (48.1%)	15 (18.9%)
Facebook	259 (35.3%)	342 (46.6%)
Dropbox	16 (22.9%)	1 (1.5%)
PayPal	61 (17.5%)	140 (40.2%)
Google	10 (10.5%)	11 (11.6%)
Ebay	8 (8.9%)	9 (10.0%)

Table 5.1: String and code obfuscation in phishing pages.

Table 5.1 presents the percentage of phishing pages that contain obfuscation indicators. Since we focus on strong and well-known indicators only, the results are likely to represent a lower bound of code obfuscation in phishing. For example, we find that some Adobe phishing pages adopt php script "action.php" for login forms. The script is invoked from a php file stored in a relative path. Automated analysis of php code (in a relative path) to detect obfuscation is a challenging problem itself.

5.3.3 Feature Engineering And Building A ML Classifier

After understanding the common evasion techniques, we now design a new machine learning based classifier to detect squatting phishing pages. The key is to introduce more reliable features. Below, we first introduce our feature engineering process and then we train the classifier using the ground-truth data obtained from PhishTank. Finally, we present the accuracy evaluation results.

Based on the analysis in §5.3.2, we show that visual features, text-based features and javascript based features can be evaded by obfuscations. We need to design new features to compensate for existing ones. More specifically, we are examining squatting domains that are already suspicious candidates that attempt to impersonate the target brands. Among these suspicious pages, there are two main hints for phishing. First, the page contains some keywords related to the target brands either in the form of plaintext, images, or dynamically generated content by Javascripts. Second, the page contains some "forms" to trick users to

enter important information. For example, this can be a login form to collect passwords or payment forms to collect credit card information.

To overcome the obfuscations, our intuition is that no matter how the attackers hide the keywords in the HTML level, the information will be *visually* displayed for users to complete the deception. To this end, we extract our main features from the *screenshots* of the suspicious pages. We use optical character recognition (OCR) techniques to extract text from the page screenshots to overcome the text and code level obfuscations. In addition, we will still extract traditional features from HTML considering that some phishing pages may not perform evasion. Finally, we consider features extracted from various submission "forms" on the page. All these features are independent from any specific brands or their original pages. This allows the classifier to focus on the nature of phishing.

Image-based OCR Features. From the screenshots, we expect the phishing page to contain related information in order to deceive users. To extract text information from a given page screenshot, we use OCR (Optical character recognition), a technique to extract text from images. With the recent advancement in computer vision and deep learning, OCR's performance has been significantly improved in the recent years. We use the state-of-the-art OCR engine Tesseract [8] developed by Google. Tesseract adopts an adaptive layout segmentation method, and can recognize texts of different sizes and on different backgrounds. According to Google, the recent model has an error rate below 3% [9], which we believe this is acceptable for our purpose. By applying Tesseract to the crawled screenshots, we show that Tesseract can extract text such as "paypal" and "facebook" directly from the logos areas of the screenshots. More importantly, from the login form areas, it can extract texts such as "email" and "password" from the input box, and even "submit" from the login buttons. We treat the extracted keywords as *OCR features*.

Text-based Lexical Features. We still use text based features from HTML to complement OCR features. To extract the lexical features, we extract and parse the text elements from the HTML code. More specifically, we focus on the following HTML tags: h tag for all the texts in the headers, p tag for all the plaintexts, a tag for texts in the hyperlinks, and title tag for the texts in the title of the page. We do not consider texts that are dynamically generated by JavaScript code due to the high overhead (which requires dynamically executing the javascript in a controlled environment). We treat these keywords as *lexical features*.

Form-based Features. To extract features from data submission forms, we identify forms from HTML and collect their attributes. We focus on 4 form attributes: type, name, submit and placeholder. The placeholder attribute specifies a short hint for the input box. Often cases, placeholder shows hints for the "username" and "password" in the phishing pages, *e.g.*, "please enter your password", "phone, email or username". The name attribute specifies the name of the button. We treat the texts extracted from the form attributes as features. We also consider the number of forms in the HTML document as a feature.

Features that We Did Not Use. Prior works have proposed other features but most of which are not applicable for our purpose. For example, researchers of [12, 58, 208] also considered OCR and lexical features, but the underlying assumption is that phishing sites share a high level similarity with the real sites (visually or keyword-wise). However, this assumption is not necessarily true given the evasion techniques and the large variances of phishing pages (§5.3.2). In addition, Cantina [216] and Cantina+ [208] propose to query search engines (*e.g.*, Google) using the keywords of the suspicious web pages to match against the real sites. However, these features are too expensive to obtain given the large scale of our dataset. To these ends, the features we chose in this dissertation (*e.g.*, keywords from logos, login forms, and other input fields) are lightweight and capture the essentials of a phishing page which are difficult to tweak without changing its impression to a user.

Discussions on the Feature Robustness. So far, we haven't seen any real-world phishing pages that attempt to evade the OCR engine. Future attackers may attempt to add adversarial noises to images to manipulate the OCR output. However, technically speaking, evading OCR features are difficult in the phishing contexts. First, unlike standard image classifiers that can be easily evaded [34, 75, 80, 118, 133, 154, 211], OCR involves a more complex segmentation and transformation process on the input images before the text extraction. These steps make it extremely difficult to reverse-engineer a blackbox OCR engine to perform adversarial attacks. A recent work confirms that it is difficult to evade OCR in a blackbox setting [177]. Second, specifically for phishing, it is impossible for attackers to add arbitrary adversarial noises to the *whole screenshots*. Instead, the only part that attackers can manipulate is the actual images loaded by the HTML. This means texts of the login forms and buttons can still be extracted by OCR or from the form attributes. Finally, for phishing, the key is to avoid alerting users, and thus the adversarial noise needs to be extremely small. This further increases the difficulty of evasion. Overall, we believe the combination of OCR features and other features helps to increases the performance (and the robustness) of the classifiers.

After the raw features are extracted, we need to process and normalize the features before used them for training. Here, we apply NLP (natural language processing) to extract meaningful keywords and transform them into training vectors.

Tokenization and Spelling Checking. We first use NLTK [25], a popular NLP toolkit to tokenize the extracted raw text and then remove the stopwords [7]. Since the OCR engine itself would make mistakes, we then apply spell checking to correct certain typos from OCR. For example, Tesseract sometimes introduces errors such as "passwod", which can be easily corrected to "password" by a spell checker. In this way, we obtain a list of keywords for each page.

Algorithm	False Positive	False Negative	AUC	ACC
NaiveBayes	0.50	0.05	0.64	0.44
KNN	0.04	0.10	0.92	0.86
RandomForest	0.03	0.06	0.97	0.90

Table 5.2: Classifiers' performance on ground-truth data.

Feature Embedding. Next, we construct the feature vector. For numeric features (*e.g.*, number of forms in HTML), we directly append them to the feature vector. For keyword-related features, we use the frequency of each keyword in the given page as the feature value. During training, we consider keywords that frequently appear in the ground-truth phishing pages as well as the keywords related to all the 766 brand names. The dimension of the feature vector is 987 and each feature vector is quite sparse.

Classifiers. We tested 3 different machine learning models including Naive Bayes, KNN and Random forest. These models are chosen primarily for efficiency considerations since the classifier needs to quickly process millions of webpages. Table 5.2 shows the results of 10-fold cross-validation. We present the false positive rate, false negative rate, area under curve (AUC) and accuracy (ACC). We show that Random Forest has the highest AUC (0.97), with a false positive rate of 0.03 and a false negative rate 0.06. The classifier is highly accurate on the ground-truth dataset.

5.3.4 Ground-Truth Evaluation

We apply the Random Forest classifier to the collected web and mobile pages from the squatting domains we detected [193][3]. As shown in Table 5.4, the classifier detected 1,224 phishing pages for the web version, and 1,269 phishing pages for the mobile version. Comparing to the 657,663 squatting domains, the number of squatting phishing pages are relatively small (0.2%).

Manual Verification. After the classification, we manually examined each of the detected phishing pages to further remove classification errors. During our manual examination, we follow a simple rule: if the page impersonates the trademarks of the target brands and if there is a form to trick users to input personal information, we regard the page as a phishing page. As shown in Table 5.4, after manual examination, we confirmed 1,175 domains are indeed phishing domains. Under these domains, there are 857 web phishing pages which count for 70.0% of all flagged web pages by the classifier. In addition, we confirmed even more mobile phishing pages (908) which count for 72.0% of all flagged mobile pages.

Reporting Phishing Websites. In September 2018, we checked PhishTank, eCrimeX and VirusTotal again. Among the 1,175 verified squatting domains, 1,075 of them are still online, and only 60 (5.1%) of them are blacklisted. We then reported the rest 1,015 phishing

[3]We refer the readers to the paper [193] for complete details.

websites to Google safe browsing (under VirusTotal). Like most blacklists, Google safe browsing does not support batch reporting, and has strict rate limits and CAPTCHAs to prevent abuse. We submitted the malicious URLs one by one manually.

Blacklist	PhishTank	VirusTotal	eCrimeX	Not Detected
Domains	0 (0.0%)	100 (8.5%)	2 (0.2%)	1,075 (91.5%)

Table 5.3: Detected squatting phishing pages by popular blacklists. VirusTotal contains 70+ blacklists.

Evading Popular Blacklists. The phishing pages detected by our system are largely previous-unknown phishing pages. To examine how likely they can evade existing blacklist, we perform a quick test. As shown in Table 5.3, we run the list of verified squatting phishing domains against several popular phishing blacklists in May 2018. First, we checked the PhishTank and find that only 2 of our squatting phishing domains have been reported (0.1%). Then we query VirusTotal [10], which contains over 70 different blacklists. These 70 blacklists collectively marked 110 (8.2%) of squatting phishing domains. Finally, examine eCrimeX [2], a phishing blacklist maintained by the Anti Phishing Work Group (APWG). Their phishing URLs are gathered from a large number organizations around the globe. Through collaboration, we obtained 335,246 phishing URLs reported during April 2017 to April 2018. In total, eCrimeX marked 4 squatting phishing domains (0.2%). Collectively these blacklists only detected 8.4% of the squatting phishing pages, which means 91.5% of the phishing domains remain undetected for at least a month. As a comparison, a recent study [78] shows that phishing pages hosted on *compromised* web servers typically last for less than 10 days before they are blacklisted. This suggests that squatting phishing domains are much more difficult to detect.

Type	Squatting Domains	Classified as Phishing	Manually Confirmed	Related Brands
Web	657,663	1,224	857 (70.0%)	247
Mobile	657,663	1,269	908 (72.0%)	255
Union	657,663	1,741	1175 (67.4%)	281

Table 5.4: Detected and confirmed squatting phishing pages.

5.4 How Browsers Defending Against Phishing Pages Performing Evasive Domain Names

For evasive domain names, instead of building a machine learning classifier, we seek to understand how robustness for the existing browsers defending against the phishing pages that perform evasion in the domain names, in particular for the homograph IDN domain [86].

Considering the potential risk of homograph IDNs, browsers have started to implement defense mechanisms. In this section, we investigate how major browsers construct their IDN defense policies, and build testing cases to systematically evaluate the effectiveness of their policies. This current section (Section 5.4) will be focused on browser policies and constructing test cases. In Section 5.5, we will present our testing results on the latest browsers and their historical versions, and examine the longitudinal browser policy changes.

5.4.1 Browsers' IDN Policies

To understand how major browsers handle IDNs, we first select a set of popular browsers based on their current and historical market shares [145, 179, 200]. We choose Chrome, Safari, Firefox, IE, and Windows Edge to analyze their publicly-available documentations and compare their *claimed* IDN policies. Table 5.5 summarizes the main policies across browsers.

Chrome. For Chrome, we summarize the main policies related to IDN homograph and omit those related to IDNA implementations [73]. First, Chrome defines policies to allow and disallow certain characters from different Unicode scripts to be mixed in a single domain name (P1 and P2). For example, Latin, Cyrillic or Greek characters cannot be mixed with each other; Latin characters in the ASCII range can be mixed only with Chinese, Japanese and Korean; Han can be mixed with Japanese and Korean. Second, Chrome will compare the "skeleton" of the IDN with top domain names[4] (and domain names recently visited by the user). The skeleton is computed, for example, by removing diacritic marks. (*e.g.*, www.googlé.com with "é" replaced by "e"). This rule is called *skeleton rule* (P3). Third, if an IDN contains mixed scripts and also confusable characters or dangerous patterns, Chrome will display Punycode (P4). Finally, P5 is used on domain names of whole-script confusables. Whole-script confusable means the domain name does not have mixing characters from different scripts. Instead, all the characters are from a single script, but can be confusable with ASCII letters. In this case, Chrome will check if the TLD is allowable. For example, attackers may construct apple.com using only Cyrillic characters. Here, TLD ".com" is not Cyrillic, and thus it is not allowed.

Firefox. Firefox's policies [144], as shown in Table 5.5, are different from those of Chrome. For example, Firefox does not have the skeleton rule to compare the IDN with popular domain names. Before 2012, Firefox only allowed IDNs with certain TLDs to be displayed in Unicode. However, as ICANN opens more TLDs, this approach becomes burdensome because Firefox has to constantly update the list. After 2012, Firefox added the mixing script check, which is similar to that of Chrome, but has different allowed/disallowed script combinations. For example, Firefox allows "Latin + Han + Hiragana," "Latin + Han + Bopomofo," "Latin + Han + Hangul," and "Latin + any single other Recommended/Aspirational scripts except

[4]Chrome has a hard-coded list of top domain names. According to the source code of Chromium, there are 5001 top domain names on the list.

Policy	Chr.	Fir.	Saf.	IE	Edge
P1: Unicode script mix (blocked)	✓	✓			✓
P2: Unicode script mix (allowed)	✓	✓		✓	✓
P3: Skeleton rule (top domain)	✓				✓
P4: Confusable chars (blocked)	✓				✓
P5: Whole-script + allowed TLD	✓	✓			✓
P6: Unicode script (allowed)			✓		

Table 5.5: The claimed policies of different browsers based on public documentations.

Category	Policy	Example IDNs	# Testing IDNs
Test-1	P1	ãȧ.com, aaȧ.com	1,000
Test-2	P2	cɯ.com, c0.com	1,442
Test-3	P5	xjö.net, jx.com	997
Test-4	P4	b−.com, b∕b.com	1,090
Test-5	P3	healtn.com, jetblue.com	978
Test-6	P6	i.com, h.com	166
Test-7	P4	fastcompany.com, gamèrsky.com	493
Test-8	P3	usa.gov, princetoŋ.edu	1,200
Test-9	P1	ḿȯȯ.bot, i.d.ȩ.ȯ.com	873
Test-10	P1	àl.net, ṽçǵ.com	880
Total			9,119

Table 5.6: Testing cases and their related browser policies (the list of browser policies is in Table 5.5).

Cyrillic or Greek."

Safari. Based on a security update in 2016 [18], Safari maintains a list of allowed scripts. Any IDNs containing scripts that are not on the allowed will be displayed in Punycode. This list has actively excluded Latin lookalike scripts such as Cherokee, Cyrillic, and Greek.

Internet Explorer (IE). IE only allow ASCII characters to be mixed with a certain set of scripts [134].

Microsoft Edge. Edge has two generations. For the legacy Edge (based on EdgeHTML), we cannot find any public documentations on their IDN policies. The new generation of Edge is based on Chromium. We assume Edge Chromium has the same policy as Chrome (as marked in Table 5.5) and will run experiments in Section 5.5 to validate this assumption.

User Configurations. Certain browsers allow user configurations. For example, Firefox allows users to disable IDN altogether and always display Punycode [70]. For IE, user-configured "accept language" can affect the IDN display. For example, if an IDN contains characters that are not part of the "accept language," IE will display the Punycode [134].

5.4.2 Building Testing Cases

Next, we design testing cases to systematically evaluate browsers' IDN policies. We focus on two main aspects: 1) we design cases to test the implementation correctness of the rules in the claimed policies; 2) we design cases that are likely to bypass known policies. We seek to test a number of browsers (of different versions, across different platforms) to understand how the policy implementations evolve over time.

We develop 10 categories of testing cases, as shown in Table 5.6. The testing cases cover all the policies in Table 5.5. We do not plan to test user configurations since they depend on user preference. For each category, we try to construct about 1,000 testing IDNs[5]. After generating the domain names, we then remove the *live domains* to 1) avoid disruptions to these live domains; 2) to improve the speed and stability of large-scale testing (*i.e.*, live domains take a much longer time to resolve and display). We have verified that all browsers will execute the same policies regardless if the domain is live or not. In total, we obtain 9,119 IDNs.

Testing the Claimed Policies Directly. As shown in Table 5.6, categories 1–6 are designed to directly test the claimed policies to examine if they are implemented correctly. The constructed testing cases are focused on testing the claimed rules instead of aiming for high-quality impersonations.

- **Category 1.** Most browsers do not allow the mixing between Latin, Cyrillic and Greek characters (P1). To test this rule, we construct IDNs that consist of mixing characters randomly sampled from Latin, Cyrillic, and Greek Unicode blocks (17 blocks in total). We randomly sample 2 characters from each of the 17 Unicode blocks to generate the 1,000 mixing-script IDNS (covering 4 types of combination: Latin + Cyrillic, Latin + Greek, Cyrillic + Greek, Latin + Cyrillic + Greek).
- **Category 2.** Chrome and Firefox claim to allow Latin characters to be mixed with Chinese, Japanese and Korean (CJK) characters (P2). However, it is not clear if other combinations are allowed. We construct IDNs that mix Basic Latin and 172 other non-CJK Unicode blocks. By randomly sampling 3 characters per block, we mix them to generate 1,442 testing IDNs.
- **Category 3.** This category is designed for whole-script confusable domain names, *i.e.*, all the characters are from a single look-alike script without any mixing (P5). For example, xn–80ak6aa92e.com (apple.com) is all Cyrillic characters but the TLD (com) is not Cyrillic, and thus should be blocked. To test this rule, we construct 997 IDNs by combing whole-script confusables from Cyrillic and common TLDs (3 ASCII TLDs .com, .net, .org and 2 IDN TLDs . , .).

[5]When constructing IDNs for a given category, we try to identify all the relevant Unicode blocks, and then randomly sample the same number of characters from each block. Sometimes, we do not get exactly 1,000 IDNs because 1,000 cannot be divided evenly by the number of related Unicode blocks or there are not enough qualified characters.

- **Category 4.** Chrome claims that if the IDN matches some dangerous patterns, it will display Punycode. The dangerous patterns include certain Japanese characters that can be mistaken as slashes, certain Katakana and Hiragana characters that look like each other. It is also not allowed to use U+0307 (dot above) after 'i', 'j', 'l' or dotless 'i' (U+0131). We construct 1,090 testing cases accordingly.
- **Category 5.** This category is used to test the skeleton rule (P3). Chrome checks whether the domain name looks like one of the top-ranked domains, after mapping each character to its spoofing skeleton. Chrome uses Unicode official confusable table [197] and 31 additional confusable pairs to map a spoofing character to its ASCII skeleton. We use the same confusable pairs to construct 978 homograph IDNs.
- **Category 6.** Safari claims to only allow scripts that do not have ASCII look-alikes (P6). For this category, we randomly pick characters from Cyrillic, Greek and Cherokee Unicode blocks (without any mixing) to form 166 IDNs.

Testing to Bypass the Claimed Policies. Next, we assume all the claimed policies are correctly implemented. Under this assumption, we construct IDNs that are likely to bypass existing policies. For these testing cases, we explicitly construct *homograph IDNs* that impersonate target domains.

- **Category 7.** Given the possibility that the Unicode confusable table used by browsers is incomplete, we test to use a more comprehensive confusable database provided by researchers [183]. We generate 493 homograph IDNs to impersonate 200 domains sampled from Alexa top 10K [1].
- **Category 8.** The skeleton rule is currently applied to 5K popular domain names. However, many important websites are not necessarily "popular" (*e.g.*, based on traffic volume). For example, websites of governments, military agencies and educational institutions all have a high phishing value, but are not ranked to the top. As such, we construct homograph IDNs for .gov, .mil and .edu domain names.
- **Category 9.** In this category, we test whole-script confusables beyond Cyrillic. We use extended sets of confusable scripts to construct homograph IDNs without mixing. We randomly sample 200 target domains from Chrome's top domain list, and generate up to 5 all-substitution homograph domains for each target domain. We also keep the original TLDs unchanged.
- **Category 10.** Most browsers prohibit the mixing between Latin, Cyrillic and Greek. However, each script has multiple Unicode blocks, and it is not clear we can mix different blocks under the same script. For example, Latin has at least 9 blocks including Basic Latin, Latin Extended-A to E (5 blocks), IPA Extensions, Latin Extended Additional, and Latin-1 Supplement. We want to understand, for instance, if Latin Extended-A and Latin Extended-B can be mixed. We construct 880 IDNs using characters within Latin look-alike Unicode blocks. All the IDNS are homograph domains impersonating 200 domain names randomly sampled from Alexa top 1 million list [1].

Desktop (Total # of Versions)	Version Range
Chrome (21)	43.0 – 81.0
Firefox (15)	54.0 – 75.0
Safari (4)	10.0 – 13.0
Edge Legacy (4)	15.0 – 18.0
Edge Chromium (2)	80.0 – 81.0
IE (4)	8.0 – 11.0
Mobile (Total # of Versions)	**Version Range**
Android Chrome (7)	5.0 – 9.0
iOS Safari (13)	10.2 – 13.2

Table 5.7: Tested browsers and their versions.

Due to the space limit, we make the list of testing IDNs available under an anonymous link[6].

5.5 Browsers Measurement Methods and Results

With the testing cases, we present our empirical experiments on major browsers and their historical versions to understand the effectiveness of IDN policies. We test historical versions for two reasons. First, it helps us to understand how different policies and their implementations evolve over time. Second, many users and organizations are still using outdated browsers [32] – their IDN policies are worth investigating.

We design experiments to answer four key questions. *First*, how well do browsers enforce known IDN policies? *Second*, how effective are existing policies in detecting homograph IDNs that impersonate target domains? *Third*, how are browser defenses changing over time? *Fourth*, are the policies' weaknesses exploited in practice?

5.5.1 Testing Platform and Methods

Browser Versions. We performed the experiments during April – May in 2020. The browser versions are shown in Table 5.7. We have primarily focused on Chrome, Safari, Firefox and Microsoft Edge. Note that Microsoft has stopped IE at its last version at v11.0 in 2016 [135], and continued with the new Microsoft Edge browser. For completeness (and considering users may use outdated browsers [32]), we have tested the legacy versions of IE too. For mobile browsers, we have tested Android Chrome and iOS Safari across their latest and historical versions.

Regarding the historical versions, we did not start from a browser's first version because

[6] https://www.dropbox.com/s/8wuy1u8gsjro9pc/

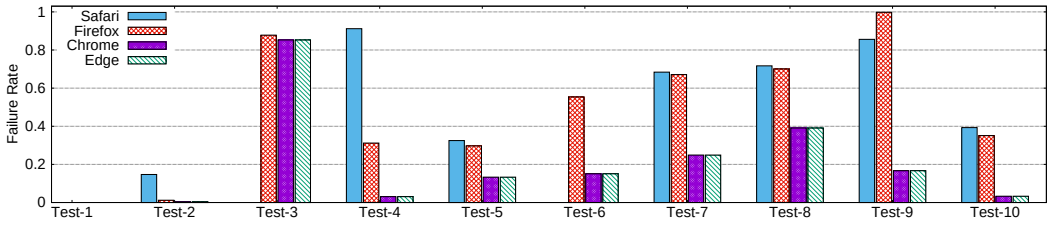


Figure 5.2: Failure rates of the 10 testing categories for the latest version of four browsers.

most browsers did not support internationalized domain names in the beginning. Without IDN support, there is no point to test IDN defense policies.

Testing Method. We run black-box testing on each browser. By loading the testing cases, we examine whether the browser displays the Unicode or the Punycode. We instrument the browsers to load the testing IDNs sequentially, and record a video to capture the screenshots of the browser. We choose to record a video (continuously) instead of taking screenshot images one by one to speed up the testing. In addition, video recording is easier to implement across platforms. To help with our post-analysis, between two consecutive IDNs, we load a special delimiter "aaaaaa——{index}" into the browser. This delimiter helps to accurately slice the video frames and map them to the IDN test cases.

A key challenge to fully automate the testing is to configure the right environment for the browsers. For example, we need a different desktop OS (*e.g.*, Windows, Linux) and mobile OS (*e.g.*, Android, iOS) for the tests. More importantly, given the need to test historical versions, we need the right *legacy OS versions* to support outdated browsers. To resolve this problem, we used a testing framework called LambdaTest [114]. LambdaTest provides remote Selenium for desktop browsers and Appium for mobile browsers, which can be controlled by our testing scripts. We can specify the operating system name and version via a configuration file ahead of testing to set up the testing environments in the cloud.

Extended Testing vs. Simplified Testing. We divide our testing into two phases. First, we run an *extended test* on the latest versions of browsers, using all 9,119 testing cases constructed in Section 5.4.2. The goal is to understand the effectiveness of the current IDN policies. This test covers Chrome 81.0, Firefox 75.0, Safari 13.0, Edge Chromium 81.0, Android Chrome 9.0, and iOS Safari 13.2. This test does not cover IE or Edge Legacy since Microsoft has chosen Edge Chromium over the other two. We consider IE or Edge Legacy as historical browsers.

Second, for all other historical versions, we run a *simplified test* considering the scalability requirement for covering a large number of browsers versions on different platforms. We sample about 10% of the testing cases for each category. For certain categories, the sampling rates are slightly higher than 10% in order to cover all the relevant Unicode blocks. The simplified test in total covers 1,027 IDNs.

Video Analysis. The video analysis aims to determine whether a given IDN is displayed

	Chrome	Firefox	Safari	Edge
Unicode	1,812	3,965	3,813	1,812
Failure Rate	19.87%	43.48%	41.81%	19.87%

Table 5.8: Testing results of the latest browsers. In total, 9,119 IDNs are tested per browser. We report the number of IDNS displayed as Unicode (*i.e.*, IDNs that browsers fail to block).

as Unicode (allowed) or Punycode (blocked) by the browser. First, we slice the video frames and map them to the specific IDN. As mentioned before, between two consecutive IDNs, we have loaded a delimiter. For example, delimiter "aaaaaa——b16" means the next video frames should be mapped to testing case #16 in category 2 (based on "b"). After slicing the video frames, we remove duplicated images based on perceptual hash (or phash) [11]. Given an image, we first crop the image to focus on the browser address bar. Then we apply OCR (Optical Character Recognition) to extract the URL in the text format from the images. We use Google's Tesseract OCR tool [74] which is known to have a good performance. If the extracted URL starts with "xn—", then we determine it is a Punycode, indicating the browser has blocked the testing IDN. We have taken additional steps to improve the accuracy of ORC, for example, by converting images into black and white, and improving the resolution of images. We randomly sampled and inspected 100 images for each browser to make sure the Punycode detection is reliable.

5.5.2 Results: Web Browsers

We start with the latest versions of web browsers. In Table 5.8, we report a *failure rate* which is the ratio of IDNs that are displayed as Unicode over all the tested IDNs. Displaying Unicode indicates that the browser has failed to block the IDN. In Figure 5.2, we show the failure rate for each testing category. Note that the failure rate has slightly different meanings for categories 1–6 and 7-10. For categories 1–6, it means the browser does not fully execute the claimed policies, which gives attackers the *opportunity* to create homograph IDNs. For categories 7-10, since all the testing IDNs are already homograph IDNs, the failure rate indicates risks more directly.

Chrome and Edge (Chromium). The first observation is Chrome and Edge have identical numbers in both Table 5.8 and Figure 5.2. This indicates Edge has the same polices as Chrome due to the use of Chromium. As such, we use Chrome as an example to discuss them together.

Table 5.8 shows that Chrome has the strictest policies compared to Firefox and Safari. Only 1,812 out of 9,119 IDNs (19.87%) are displayed in Unicode by Chrome. Noticeably, Chrome (and all other browsers) has a failure rate of 0% under category-1 (Figure 5.2). It means browsers enforced the rule to prevent the mixing of Latin, Cyrillic or Greek characters.

However, for the other nine categories, Chrome's failure rate is non-zero. The result of categories 2–6 suggests that Chrome does not fully enforce the rules as claimed. Category-3 has the highest failure rate (85.3%). It turns out that Chrome allows whole-script confusables from Cyrillic to be combined with common TLDs such as .com and .net. The other 14.7% IDNs in category-3 are blocked because they have triggered other rules (e.g., skeleton rule). The results in categories 4 and 6 indicate Chrome does not fully cover Unicode confusables in the Unicode documentation and all the ASCII look-alike scripts. Category-5 has a failure rate of 13.3% (skeleton rule), indicating the skeleton comparison cannot perfectly capture all the confusable characters in the top domains.

For categories 7–10, the results confirm that our strategies to bypass Chrome policies are largely successful. In category-7, by using a more extensive confusable character table, we can cause more failures to the skeleton rule. In category-8, we focus on target domain names that are not in the top domain list (*e.g.*, those under .edu, .gov, and .mil), and Chrome fails to capture 40% of the homograph IDNs. Certain Unicode blocks are consistently missed. For example, when using confusable characters from the "Latin Extended-A" to impersonate the .edu, .gov, and .mil domain names, the failure rate is 100%. For categories 9 and 10, while the failure rates are lower, the results still indicate there are exceptions in the current mixing rule. For example, full-Substitution with "Latin Extended-A" causes a 100% failure rate, followed by a full-Substitution with "Cyrillic." Also, certain blocks within the Latin can be mixed without alerting Chrome (*e.g.*, mixing "Latin Extended-A" and "Latin-1 Supplement").

Safari. Safari has a failure rate of 41.81% overall. Compared to Chrome, Safari does not implement as many rules. For the rule that Safari did implement (*e.g.*, the rule corresponds to category-6), Safari does not make any mistakes. In addition, Safari blocks all the IDNs in category-1 (mixing script) and category-3 (whole-script Cyrillic). This is because Safari's allowed scripts have already excluded Latin lookalike scripts such as Greek and Cyrillic. Even so, it is still feasible to create homograph IDNs to bypass Safari. As shown in Figure 5.2, Safari has a failure rate over 60% on the homograph IDNS in categories 7, 8 and 9.

Firefox. Firefox has a higher failure rate (43.48%) among tested browsers. In particular, Firefox does not implement the skeleton rule, and thus the corresponding testing cases (categories 5, 7, 8) all have relatively higher failure rates.

Case Studies. So far, we have discovered several strategies to bypass existing IDN policies. Some of the strategies are more useful than others to craft high-quality homograph IDNs. To illustrate the differences, in Table 5.9, we present example homograph IDNs crafted for Chrome, based on the mistakes Chrome made in each category (except for category-1 where Chrome has no failure). We find that it is easy to craft homograph IDNs for categories 3, 5, 7 and 8. For category-4, most of the dangerous patterns are mimicking non-English letters and symbols (such as slash). This limits the ability to generate homograph IDNs. For category-6, although we have found a large number of individual characters from different Unicode scripts missed by Chrome, it is not easy to craft high-quality homograph IDNs due to other rules (*e.g.*, non-mixing rules, skeleton rules). For categories 9 and 10, although we

Category	Example IDNs to Bypass Chrome
Test-2	iú.edu, huÍu.com
Test-3	ucla.edu, usçis.gov
Test-4	openaı.com, āi.google
Test-5	iħstagram.com, weılsfargo.com
Test-6	ţɳţ.com, ԋƅƿ.com
Test-7	tumbır.com, ħyc.gov
Test-8	defẹnse.gov, pénnsylvania.gov
Test-9	ħŷć.gov, ħŷċ.gov
Test-10	ĵęţ.com, ǰđ.id

Table 5.9: Example homograph IDNs that can bypass Chrome's policies to be displayed in Unicode.

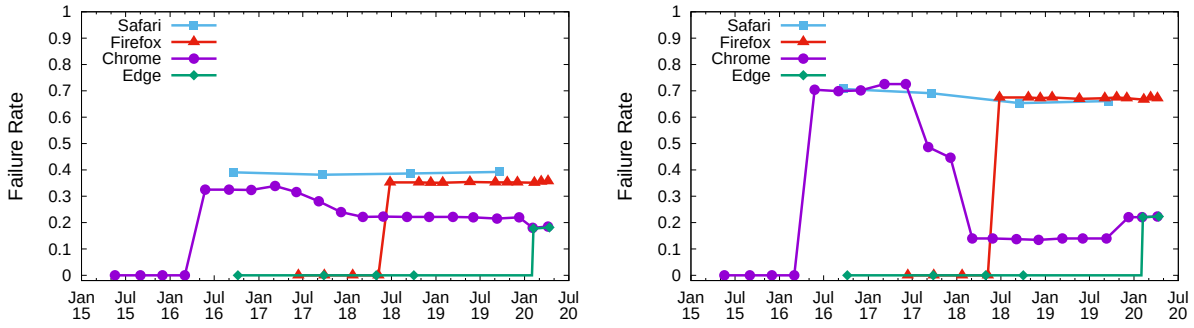


Figure 5.3: Failure rates over time for different browser versions from January 2015 to April 2020, left figure is testing categories 1–6, and right figure is testing categories 7–10.

can easily find homograph IDNs, the IDNs need to be whole-script (*i.e.*, all the characters need to be replaced), and thus might sacrifice the quality of impersonation. Overall, the exception rules identified for categories 3, 5, 7 and 8 are the most effective ways to craft homograph IDNs.

5.5.3 Results: Mobile Browsers

We perform the same analysis on the mobile browsers including Android Chrome and iOS Safari. After analyzing their latest and historical versions, we find that the results are exactly the same as the corresponding web versions (Chrome and Safari). As such, we use "Chrome" and "Safari" to represent both the web and mobile versions. Note that mobile browsers present additional challenges for users to recognize web domain names due to the limited screen size. Some mobile browsers would only display part of the URLs or even hide the whole URLs in the address bar [125, 126], which heightens the security risks. The user interface (UI) design, however, is beyond the scope of this dissertation.

5.5.4 Browser Policy Changes Over Time

Next, we analyze the historical browser versions to understand how their IDN policies change over time. Given a browser, we sort all its versions by the release dates. Then we select the most updated version for each quarter (4 quarters per year) to report their failure rates. As shown in Figure 5.3, we break down the results for categories 1–6 (Figure 5.3) and categories 7–10 (Figure 5.3) since their failure rates have different meanings. We have merged the curve for Edge Chromium and Edge Legacy since their release times do not overlap. We have also tested IE, but all the testing cases are displayed as Punycode. As such, we omit IE from Figure 5.3 for brevity.

Overall, most browsers follow a similar trend. *First*, the failure rates were initially at 0% because the browser did not support IDN yet in the early versions. All the testing IDNs are displayed as Punycode. These include Chrome browsers before version 51.0 (released in June 2016), Firefox browsers before version 61.0 (released in June 2018), and Edge browsers before 80.0 (released in February 2020). *Second*, once the browser started to support IDNs, the failure rates immediately jumped to a high level due to a lack of defense policies. *Third*, for browsers such as Chrome and Safari, their failure rates were gradually decreasing afterward as browsers added new IDN policies. For example, starting in March 2017, Chrome had a series of updates that significantly decreased the failure rate (mostly for categories 2, 5, 8, 9, and 10). In comparison, Firefox's failure rate has stayed at a similar level, indicating fewer or no updates of its IDN policies. As mentioned before, Edge changed to use Chromium in early 2020, and has followed Chrome's IDN policies since then.

One interesting observation (see Figure 5.3) is that Chrome's failure rate went higher at the end of 2019, indicating certain policies were revoked. A further inspection shows the blocking decisions on many testing cases in categories 5, 7 and 8 were reversed — the new Chrome version re-allowed certain homograph IDNs to be displayed as Unicode. These re-allowed homograph IDNs contain characters from three main Unicode blocks: "Latin Extended-A," "Latin Extended-B," and "Latin-1 Supplement". Homograph IDNs such as aŕmy.mil, yaĺe.edu, uchìcago.edu, canoń.com, and babblę.com can be displayed in the updated Chrome even though they were blocked by earlier versions. The reasons behind this are not clear. If they were not implementation errors, one possible explanation is that blocking these characters might hurt legitimate domain names with such characters.

5.5.5 Browsers vs. Real-world IDNs

The controlled experiments have shown the weaknesses of existing IDN policies. Next, we examine if such weaknesses are exploited by real-world homograph IDNs. To do so, we first search for homograph IDNs from DNS zone files and then test them against the latest browsers.

Records	Count
DNS records	347,014,213
Unique domain names	143,482,491
Unique IDNs	916,805
Homograph IDNs	1,855

Table 5.10: Analysis results of .com DNS zone file.

Dataset. We obtained the access to the .com zone file from Verisign Labs[7] in January 2020. We chose .com since it is the most popular top-level domain (TLD) where most commercial websites are registered. As shown in Table 5.10, there are 347 million DNS records in the zone file. Among them, there are 143 million unique domain names. For each domain name, we check whether it contains any character outside of the ASCII table.

In total, we find 916,805 IDNs. While the percentage is not high (0.64% of all .com domain names), the absolute number of IDNs is still nearly 1 million. We observe that most IDNs come from East Asia and Europe, which is consistent with that of a prior study [121]. We also find script mixing is common. Out of the 916,805 IDNs, 315,671 (34.4%) domain names have script mixing.

Homograph IDNs. To identify homograph IDNs, we follow a common detection method: 1) We select the domain names from Alexa top 10,000 domains [1] as the impersonation target; 2) We search for homograph candidates using a database of look-alike characters (*e.g.* "a" (U+0041) looks like "a" (U+0430)). We use a comprehensive homoglyph database from a recent work [183]; To detect homograph IDNs, we recursively replace the characters with their look-alike characters and search the modified domain name in our IDN list. If the modified domain name is in the list, we consider it as a homograph IDN.

In total, we identified 1,855 IDNs that impersonate 674 popular domain names. We have manually verified these IDNs to make sure the detection is reliable. The top five most impersonated targets were amazon.com, google.com, paypal.com, canva.com, and gmail.com. For example, xn—gmal-spa.com (gmaìl.com) impersonates gmail.com.

Browser Testing Results. We test the 1,855 real-world homograph IDNs by displaying them in recent Chrome 81.0, Safari 13.0, and Firefox 75.0. Displaying Punycode means browsers can successfully block the homograph IDN.

We find that Chrome displays Punycode for 1,189 homograph IDNs (64.1%); Safari and Firefox only display Punycode for 180 (9.7%) and 113 (6.1%) of them. Chrome's defense is stronger than that of Safari and Firefox (consistent with our controlled experiments). Even so, Chrome has missed 35.9% of the homograph IDNs (more than one third).

Note that our finding is slightly different from an earlier study from 2018 [121] which showed Chrome's defense was effective against homograph IDNs discovered at that time (100%

[7]Verisign Labs have made their datasets open to researchers: https://www.verisign.com/en_US/company-information/verisign-labs

detection rate). Our results indicate that attackers have already exploited new ways to construct homograph IDNs to bypass existing browser policies.

5.6 User Study For IDN Homograph Attacks

We have shown that web browsers cannot block all the homograph IDNs. Next, we present a user study to examine how end users perceive the homograph IDNs in web browsing. In particular, we want to compare the homograph IDNs that browsers (*e.g.*, Chrome) block and those that can bypass existing policies. We focus on Chrome in this user study because Chrome has the strictest policies compared to other browsers. Our study aims to answer three research questions:

- **RQ1:** Would users fall for homograph IDNs (i.e., incorrectly treating them as the real domain names)?
- **RQ2:** Would users have different rates of detecting IDNs that are blocked vs. not blocked by Chrome?
- **RQ3:** What factors are associated with users' rates of detecting IDNs? (association rather than causality)

5.6.1 Study Design

To answer these questions, we conducted an online experiment via Amazon Mechanical Turk (MTurk). Our study was approved by the IRB. The participation of the study was anonymous and voluntary. We also did not collect any personal identifiable information (PII) from the participants. Participants can choose to withdraw their data at anytime.

We presented the study as a generic survey on web browsing. We did not mention "security" or "phishing" in the study description to avoid priming users. Before the study started, we gave participants a short tutorial to explain what "domain name" and "browser address bar" are to ensure they can understand our terminology in the study. Upon finishing the study, we debriefed the participants by providing detailed explanations for the specific purpose of the study, and information on how homograph IDN is used for phishing[8].

Each participant was asked to browse a series of screenshots of website landing pages. As shown in Figure 5.4, a screenshot contains both the address bar[9] and the real landing

[8]After our study, we received messages from participants who thanked us for educating them about homograph IDNs.

[9]In the address bar of the screenshots, we always displayed the Unicode version of the homograph IDNs to examine how users perceive them and to fairly compare homograph IDNs missed by Chrome with the blocked ones. We wanted to understand whether the missed IDNs are more or less difficult to detect by users compared with the blocked ones in the Unicode format.

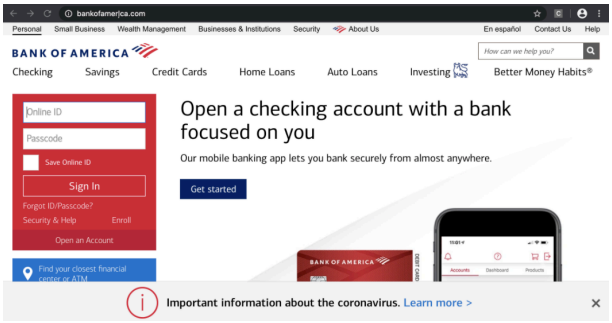


Figure 5.4: An example screenshot, which always shows the real webpage. The address bar was artificially added to display either the real domain name or a homograph IDN in Unicode (in this case, www.bankofamerļca.com). Right below the screenshot, we asked *"Is the domain name in the browser address bar* bankofamerica.com*?"* Participants can choose one of three answers: *"Yes," "No," "I can't tell."*

page. Some of these screenshots impersonated domain names with homograph IDNs (e.g., www.bankofamerļca.com in Figure 5.4), while the rest showed the real domain names. To see whether people can detect homograph IDNs, for each screenshot, we asked the participant a question about the authenticity of the website.

A key challenge was to determine how to phrase the question to the participants. At a high-level, we need to make sure users are making decisions based on the controlled information (*e.g.*, whether the domain name is a homograph IDN). This means we need to draw users' attention to the address bar. At the same time, we also wish to avoid over-priming users which will likely make the study unrealistic. In practice, users are often caught off-guard by phishing websites when they are not paying attention. Thus over-priming users could over-estimate users' ability to detect security threats [170].

Final Design. After comparing the results of the pilot studies, we decided to choose the *medium* priming level and *binary answer* (plus "I can't tell") as the final design. We asked "is the domain name in the browser address bar *[the real domain name]*?". Participants can choose one of three answers: "Yes," "No," "I can't tell." This is based on two reasons. First, we did not observe a need to use a 5-point Likert scale as the trend was the same for both conditions and using the Likert scale can complicate the tasks. People might also interpret the five levels differently. Instead, a binary answer (plus "I can't tell") can reduce the ambiguity. Second, the medium priming level (*i.e.*, mildly cuing users to check the address bar) is more suitable since our research questions are about domain names. While we use the medium priming level for our main study, the other pilot study results can serve as the lower/upper bounds of detection rates.

5.6.2 Main User Study

After determining the study design, we now introduce the setups of the main user study.

Websites. For the main user study, we use a diverse set of 90 websites. Out of the 90 target websites, 45 were from the Chromium top domain list (*i.e.*, "popular"), and the other 45 were not on the list (*i.e.*, "unpopular"). We select these websites from five common website categories (18 websites per category): "Shopping," "Banking," "Social Networking," "Education," and "Government & Military."

For each target website, we can choose to display the real domain name in the address bar of the screenshot ("Real"). We can also choose to display the homograph IDN to impersonate it. We consider two types of homograph IDNs: one IDN that can be blocked by the latest Chrome (IDN-Block), and another IDN that can bypass Chrome's policy (IDN-Pass).

Out of the 90 websites, we set the ratio of "Real", "IDN-Block", and "IND-Pass" to be roughly 1:1:2. We included more IND-Pass domains because IDNs that can bypass Chrome's policies are less understood and studied. We covered more IDN-pass domains to better study this category. More specifically, we randomly chose 23 of the 90 websites to display the real domain names ("Real"), and select another 23 websites to display homograph IDNs that can be blocked by Chrome ("IDN-Block"). For the remaining 44 websites, we crafted homograph IDNs that would bypass Chrome's policies ("IDN-Pass"). A complete list of the websites and domain names is available via this anonymous link[10].

Factors. In addition to website category and popularity, we also considered other factors such as people's demographics (e.g., , age range, gender identity) and computing/Internet experiences (e.g., , years of using web browsers, computing background). These questions are included in Appendix G.

Study Process. In April 2020, we conducted a study on MTurk. Each participant examined 30 websites. More specifically, we divided the 90 websites into 3 blocks (each block has 30 websites). In each block, the mixture ratio of "Real", "IDN-Block", and "IND-Pass" was still roughly 1:1:2. We randomly assigned each participant to one of the three blocks (each participant can work on one block only). Once the block was assigned, we presented a random order of the 30 sites in the block to the participant.

To ensure the reliability of results, we randomly selected one attention check question and inserted it in a random position in the task/question list. We have two attention questions to choose from. 1) "Is the domain name shown in the browser address bar a social networking website?" The screenshot shows the webpage of the Bank of America. 2) "Is the domain name shown in the browser address bar a hospital website?" The screenshot shows the webpage of Facebook. Both questions have the obvious answer "No", and a wrong answer could indicate the participants did not pay close attention.

To attract serious workers on MTurk, we used commonly applied filters: we recruited U.S workers who have an approval rate greater than 90%, and have completed more than 50 approved tasks. Each participant was compensated $1 for their time. The participants took

[10] https://www.dropbox.com/s/z57fi5gnmeqcrim/

Domain Type	Yes	No	I can't tell
Real	1,565 (**94.6%**)	86 (5.2%)	4 (0.2%)
IDN-Block	807 (48.8%)	803 (**48.5%**)	45 (2.7%)
IDN-Pass	1,353 (42.3%)	1,768 (**55.2%**)	79 (2.5%)

Table 5.11: Correct answer rates in the main study (6,510 answers): 94.6%, 48.5%, 55.2% for real, IDN-Block, IDN-Pass.

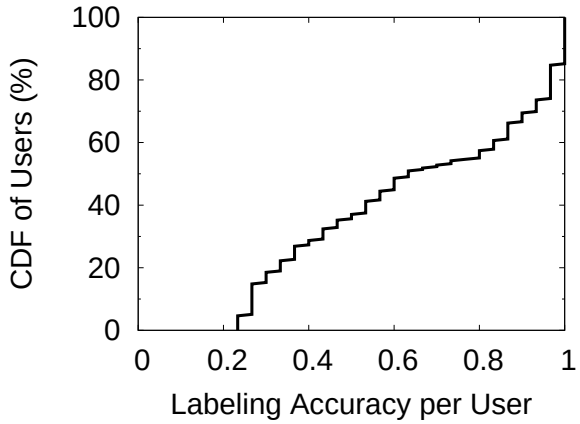


Figure 5.5: Cumulative distribution function (CDF) of labeling accuracy for each user.

8 minutes on average to complete the study. The compensation was about $8 per hour. Each worker can only participate in the study once. Pilot study participants were not allowed to take part in the main study, which had a total of 325 participants. After removing incomplete submissions and those who failed the attention check, we had 217 valid participants with 6,510 answers.

5.6.3 Overall Results

Table 5.11 shows the overall results of the main study. The results were consistent with the pilot studies. When the domain name was real, 94.6% of the answers were correct (by answering "YES"). In comparison, when the domain name was homograph IDN, only 55.2% of the answers were correct under IDN-Pass, followed by 48.5% under IDN-Block.

This result answers RQ1: our participants fell for a large percentage of homograph IDNs. We also examined how well individual participants correctly labeled the authenticity of websites based on the domain names. Figure 5.5 shows the cumulative distribution function (CDF) of each participant's labeling accuracy (based on the 30 websites the user has examined). All participants had an accuracy above 20%, and a small portion (15%) of them had a 100% accuracy. However, about half of the participants had an accuracy below 60%. Overall, the results suggest that the majority of users will struggle in correctly identifying homograph IDNs.

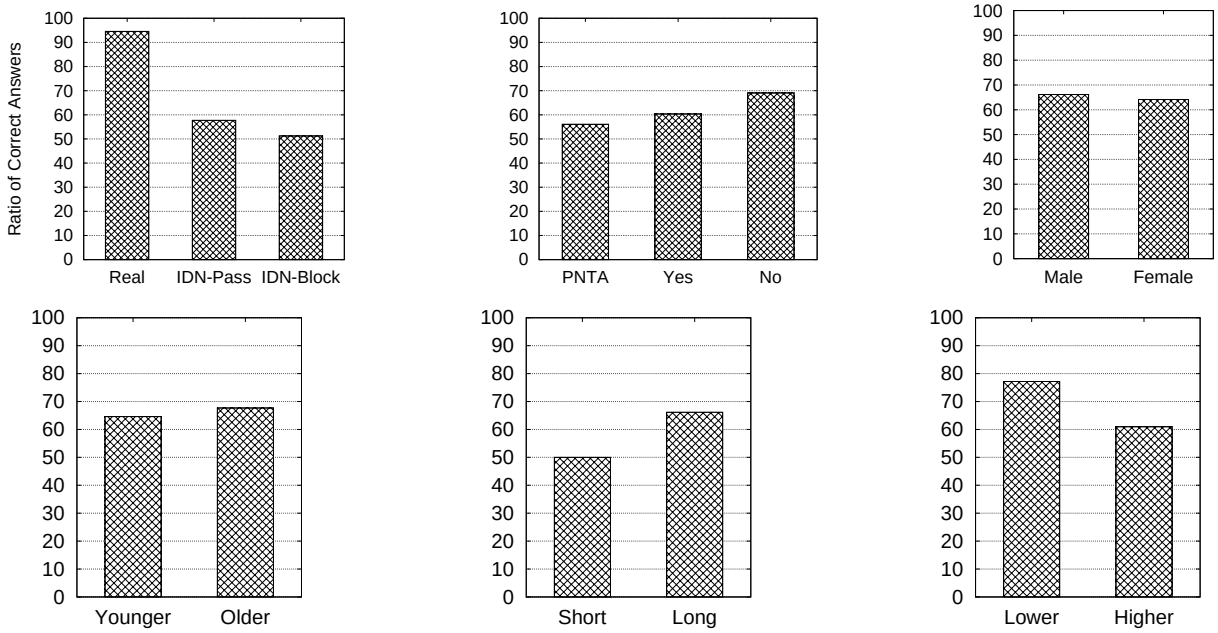


Figure 5.6: The percentages of correct answers for different groups. We include the factors that have statistical significance in Table 5.12. "PNTA" under computing background stands for "Prefer not to answer."

To answer RQ2, we then performed pair-wise comparisons between these three conditions using Chi-square tests with a Bonferroni correction (the adjusted p value threshold is .01). We found that the differences among these conditions were statistically significant: the correct answer rates for Real vs. IDN-Block ($\chi^2 = 859.3$, $p < 0.001$), Real vs. IDN-Pass ($\chi^2 = 782.7$, $p < 0.001$) and IDN-Block vs. IDN-Pass ($\chi^2 = 19.6$, $p < 0.001$). Comparing IDN-Block and IDN-Pass, we found that homograph IDNs blocked by Chrome were more deceptive (lower correct answer rate) than those not blocked.

However, it is alarming that 45% of un-blocked domain names (IDN-pass) were mistaken by our participants as real sites. Thus, they pose a substantial issue as about half of the times people fell for homograph IDNs not caught by Chrome.

5.6.4 Regression Analysis

To answer RQ3, we further analyzed the factors associated with user performance in detecting IDNs. We used the dataset of 6,510 answers and conducted logistic regression analyses in R to predict a binary outcome: whether the authenticity of a website domain name was correctly labeled by a user (*i.e.*, correct answers for Real and IDN-Block/Pass are "Yes" and "No," respectively). Table 5.12 shows the regression results.

Variable	Coefficient	P-Value
Domain Type: base = IDN-Block		
IDN-Pass	0.884	**0.006**
Real	3.441	**<0.001**
Website Category: base = Banking		
Education	0.415	0.199
Government & Military	0.301	0.287
Shopping	0.465	0.109
Social networking	0.496	0.088
Website Popularity: base = Popular		
Unpopular	-0.289	0.288
Browser experience: base = Short (<= 3 Yr)		
Long (> 3 Yr)	0.450	**0.001**
Computer background: base = NO		
YES	-0.371	**<0.001**
Gender: base = Female		
Gender: Male	0.235	**<0.001**
Age: base = Younger (<= 39)		
Age: Older (> 39)	0.133	**0.044**
Education: base = Lower (< Bachelor's)		
Higher Edu (Bachelor's or higher)	-0.823	**<0.001**

Table 5.12: Logistic regression results: using website and user factors to predict whether the authenticity of a website domain name was correctly labeled by a user.

Predictor Variables. The independent variables or predictors were all categorical variables, including the domain type, website category and popularity as well as people's demographics and computing experience.

We had three predictors related to websites. First, the domain types included Real, IDN-Pass and IDN-Block. We used IDN-Block as the baseline. Second, we had five website categories and hypothesized that the website category may affect users' judgment of the website authenticity. For example, users might be more likely to check the authenticity of banking websites than education websites. As such, we used banking websites as the baseline. Third, for website popularity, we hypothesized that users may perform better on popular websites since they might be more familiar with those. Thus, we used popular websites as the baseline.

We had five predictors related to users. First, for users' years of experience using web browsers, we converted this variable to a binary variable: "short" and "long" using 3-year as a threshold. We chose this threshold by examining the sign of the regression coefficients of the original levels and found that 3-year was the level where the sign changed. To simplify our analysis, we applied this method in converting other user-related multi-level ordinal predictors to binary variables. We hypothesized that users with a long experience with web

browsing may perform better in detecting IDNs. The second and third variables were users' computing background and gender identity. The fourth variable was age level, and we used a threshold of 39 to divide users into younger and older categories. The last user variable is education level, and we used "Bachelor's degree" to divide users into two levels.

Result Interpretation. As shown in Table 5.12, several factors were significantly correlated with users' performance in detecting IDNs. These results answer RQ3. Overall, we found that domain types and user-related factors were significantly associated with users' performance whereas website category and popularity were not. As a reference, in Figure 5.6, we further illustrate the raw percentage of correct answers for factors that have statistical significance.

First, the *domain type* results imply that participants were significantly more likely to label the real domains and IDN-Pass domains correctly compared to the baseline (IDN-Block). The result is consistent with that in Table 5.11. More specifically, Real has a β estimate of 31.23, which means the odds of labeling real domains correctly is $exp(3.441) = 31.23$ times of that of labeling IDN-Block correctly. Similarly, the odds of labeling IDN-Pass correctly is $exp(0.884) = 2.42$ times of that of labeling IDN-Block correctly. These results further confirm that Chrome indeed blocked the homograph IDNs that are more deceptive to users than IDNs that were not blocked (IDN-Pass). This does not necessarily mean IDN-Pass is safe for users. As discussed in Section 5.6.3, homograph IDNs that bypassed Chrome policies are also highly deceptive. Unlike domain type, website category and popularity were not found to be significantly associated with user performance.

Second, we found several user factors were significantly correlated with correctly labeling the website authenticity. For example, the odds of correct labeling for users with a longer (3-year) web browsing experience is $exp(0.450) = 1.59$ times of that of users with a shorter experience. Similarly, users' frequency of visiting the five categories of sites were also significantly and positively correlated with user performance. Male participants did better in correctly labeling the domain names. However, as shown in Figure 5.6, the performance difference between male and female participants was rather small (but statistically significant). Older participants seemed to perform better than their younger counterparts. Again the difference was small but statistically significant.

Third, perhaps counter-intuitively, computing background and educational level were also significantly correlated with user performance albeit negatively. As shown in Figure 5.6, the differences were relatively small (but statistically significant). Users with a higher educational level or computing background seemed to perform worse. While we do not know the reason, one plausible explanation could be that they were overly confident about their knowledge/skills and overlooked the IDNs. Future research can investigate the reason(s).

Limitations of The User Study. Our user study has many limitations. First, we cannot accurately measure user attention. Our inclusion of an attention check ameliorates this limitation. Future work could consider using eye-tracking but it is hard to deploy via

MTurk. Second, there were still differences between our study setup and real-world web browsing. In particular, we showed a screenshot of a target website, and thus participants cannot interact with the websites. The non-interactive screenshots helped to protect users and also allowed us to focus on the domain names rather than other user strategies. We also reminded our participants to pay attention to domain names, while in practice, users are likely to make more mistakes if they are not reminded (per the results of pilot studies). Finally, our user study only examines user perception of the authenticity of websites (domain names), which is only the first step in web phishing. Future work can study how IDNs affect their follow-up actions such as login.

5.7 Discussion

IDN Homograph in Email and Social Network Services. Email systems and social network platforms are also popular channels to disseminate phishing messages. In these applications, IDN homograph can be used to deceive users too. We briefly investigated popular email and social network services regarding their IDN policies. Our overall observation was that most services had not established effective IDN policies. Due to space limit, we briefly summarize our findings and leave the experiment details to Appendix-J and K.

For email services, we looked into Gmail, iCloud and Outlook. As of May 2020, we tested to see if homograph IDNs (that impersonate popular domain names) can be displayed on email clients in Unicode. For this test, we need to register the homograph IDNs and set up the DNS records (we registered 3 IDNs, and details are in the Appendix). For email clients that supported IDN, we found the homograph IDNs were consistently displayed in Unicode. For example, Gmail (web and mobile) and iCloud (mobile) supported IDN and displayed homograph IDNs in Unicode in the email sender addresses. This means attackers can use homograph IDNs to impersonate trusted senders. This observation is true even for a homograph IDN that is blocked by web browsers. The results suggest that email clients have not yet established effective policies to address homograph IDNs.

For social network applications, we examined how homograph IDNs are displayed in messages and posts. As of May 2020, we tested Facebook, Twitter, Messenger, iMessage, and Whatsapp with homograph IDN URLs that impersonate popular brands. We found that almost all of them displayed homograph IDNs in Unicode, except for Facebook (which displayed Punycode). The result again suggests that most social network applications do not have IDN defense policies.

Responsible Disclosure. As of June 2019, we have reported our findings to the corresponding bug/security teams of Chrome, Safari and Firefox. Microsoft IE uses Chromium, and thus is also covered. So far, Chrome and Firefox have started to investigate and address the reported issues.

5.8 Conclusion

In this chapter, we perform an extensive measurement on pages that perform evasions in web content evasions and domain name evasions. By monitoring 700+ brands and 600K squatting domains for a month, we identified 857 phishing web pages and 908 mobile pages. We show that squatting phishing pages are impersonating trusted entities through all different domain squatting techniques. Squatting phishing pages are more likely to adopt evasion techniques and are hard to catch. About 90% of them have evaded the detection of popular blacklists for at least a month.

We also present a detailed analysis of browsers' defense policies against IDN homograph. Using more than 9,000 testing cases, we measure the effectiveness of IDN policies in existing web and mobile browsers and their historical versions from 2015 to 2020. We show that browsers' IDN policies are not yet effective to detect homograph IDNs. Our user studies show that the homograph IDNs that can bypass browsers' defense are still highly deceptive to users. Overall, the results highlight the need to improve the defense policies.

Chapter 6

Conclusion

6.1 Summary

While machine learning has tremendous potential as a defense against real-world security threats, understanding the capabilities and robustness of machine learning remains a fundamental challenge. This dissertation tackles problems essential to the deployment of machine learning in security applications.

Defending Against Adversarial Examples. In Chapter 3, we develop D2P, a new method to automatically generate robust adversarial examples that can survive in the physical world. With D2P, we can augment the training data and further improve the robustness of machine learning models by model retraining.

Detecting Malicious Bots. In Chapter 4, we collaborate with one security company, Radware, and develop a stream based feature encoding scheme to support machine learning models for detecting advanced bots. And we further propose ODDSto synthesis malicious bots to address the problem of limited of attacker's data.

Detecting Adaptive Attackers. In Chapter 5, we perform an extensive measurement of adaptive phishing pages, where the phishing pages that are performing evasion techniques at both the domain name and web content level. By monitoring 700+ brands and 600K squatting domains for a month, we identified 857 phishing web pages and 908 mobile pages. We also perform an extensive measurement to identify the potential attacking incidences in web browsers for defending against homograph IDN attacks. As of June 2019, we have reported these fail cases to the corresponding bug/security teams of Chrome, Safari and Firefox. So far, Chrome and Firefox have started to investigate and address the reported issues.

6.2 Lessons Learned

Leverage Domain Specific Insights. When we build a machine learning-based security application, we should leverage domain-specific insights to customize the design and further robustify machine learning models. For example, in image recognition, we observe images

lose certain features due to quantization effects, our design based on this observation. For bot detection, the design is based on the domain knowledge the benign user and bots evolve at a different rate so that we can make an assumption benign users are more stable and representative. And ODDS is based on this assumption. For phishing detection, our design is based on the observation phishing pages usually contains forms to collect credential.

Building Robust Features. Feature engineering also plays an important role in machine learning based security applications. As attackers' behaviors change over time to avoid detection, simple features or rules become obsolete quickly. When we design the features, we should design the features that are hard for attackers to modify. As shown in Chapter 5, we build visual and form features from phishing sites. They are hard to modify because no matter how attackers obfuscate the HTML content, the visual presentation of the page will still need to look legitimate to deceive users. Another example is bot detection. For feature engineering in Chapter 4, there is a feature name time gap feature, it indicates the time gap between the current request and the previous request. If the attacker is going to modify the time gap feature to evade detection, it will slow down their requests sending rates dramatically.

6.3 Implications for Our Works

Image Recognition. By automatically generating realistic adversarial examples that can survive in the physical world, we can scale up several lines of applications: (1) evaluating of the robustness of real-world computer vision applications, such as object detection systems used by self-driving cars and home-security systems; (2) improving defense methods against adversarial examples. So far, most defense methods are designed to detect *digital-domain* adversarial noises [153]. Using D2P, we can generate more *realistic* adversarial examples to assist the troubleshooting of under-trained regions and augment the training data for model retraining [167] or adversary detection [212].

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

CAPTCHA system allows us to collect the labels to train a machine learning model. However, in order to get enough coverage of data, we have to aggressively deliver CAPTCHA. Our proposed method ODDS could help 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 4.8, 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.

Phishing Website Detection. Our system SquatPhi can be used in two ways. First, any *third-party organizations* can set up a scanner to constantly monitor the squatting domains for a *broad range of brands* to capture squatting phishing domains. Crowdsourcing efforts can be introduced to speed up the manual verification process. Note that we are searching needle in a haystack by narrowing down the target from hundreds of thousands squatting domains to several hundreds phishing candidates, which are then manageable for manual investigation. Second, *individual online services* can set up their own dedicated scanner to search for squatting phishing pages that impersonate their brands. For example, Paypal can keep monitoring the newly registered domain names to the DNS to identify PayPal related squatting domains and classify squatting phishing pages. The classifier can be potentially much more accurate if it is customized for one specific brand.

To improve homograph IDN detection, one way is to add new rules to address the failure cases discovered by our experiments. For example, browsers such as Chrome can extend the list of target domains (*e.g.*, for skeleton rule), use a more comprehensive confusable table (instead of the standard Unicode confusable table), and increase the number of prohibited Unicode blocks. Even so, it is difficult for the rules to guarantee completeness. For example, the skeleton rule matches the IDNs against a list of top domain names which do not cover all the domains. To extend the list (*e.g.*, to cover all the domains), the immediate challenge is efficiency. Considering the browsers' need to make decisions in real-time, it is costly to check the visual similarity between the IDN and hundreds of millions of domain names. Improving the scalability of the skeleton matching is an open challenge for future work.

6.4 Future Research Directions

This dissertation aims to design practical, robust ML algorithms in security applications. Moving forward, we hope to broaden and deepen this investigation, extending our works to more security applications and more robust machine learning algorithms. Initially, we will focus on the following three research directions.

Physical Attacks against Real ML Systems To the best of our knowledge, although adversarial examples in physical domain are getting more and more research attentions, there are very limited reported physical attacks in the 'real world applications' (most existing works target the released image recognition models [185]). One possible reason is adversarial examples in the physical domain are already much harder than in the digital domain as shown in Chapter 3, and this reason is amplified by 'black-box setting' — most of the existing

physical attacks, including D2P, only work in the white-box setting. To raise the awareness of the security issues, designing black-box physical attacks that break real ML systems, such as a security surveillance system or a smart home system, is a potential research direction.

Cost Aware Adversarial Examples. $L1$ or $L2$ distance is the common metric to measure the quality of adversarial examples. However, in many security applications, they are not practical as in typical adversarial machine learning problems since the "small changes" defined by the distance function in the feature space do not necessarily reflect the real-world costs to attackers [182]. Use the features design in Chapter 4 as an 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. Crafting cost aware adversarial examples is a potential research direction, which considers the cost and the constraints for attackers to generate the adversarial examples.

Transfer Learning. As shown in Chapter 4, it is difficult to obtain attackers' data due to their rarity. To alleviate this problem, we had tried data synthesis and the one class detection methods, such as one-class SVM. We plan to try another possible research direction, transfer learning. In the security applications, even if attackers' labels are difficult to obtain in a domain of interest, they may be obtainable in related domains. For example, security companies monitor customers' networks to prevent attacks in the future. Although attackers' labels are difficult to obtain from a new customer's network, they may be obtained from existing customer's networks that have long been monitored. Combined with data synthesis, I hope to build a robust machine learning models to address attackers' data scarce problems.

Bibliography

[1] Alexa top 1 million websites. https://www.alexa.com/topsites.

[2] eCrimeX. https://www.ecrimex.net/.

[3] Image hashing. https://github.com/jenssegers/imagehash/.

[4] Lookout advances mobile phishing protection amidst growing enterprise risk. https://goo.gl/NzBkSN/, .

[5] PhishTank. https://www.phishtank.com/, .

[6] Cyber squatters are targeting britain's biggest banks. https://goo.gl//.

[7] NLTK: Natural language toolkit. https://www.nltk.org/.

[8] Tesseract: open source ocr engine. https://github.com/tesseract-ocr/tesseract, .

[9] Tesseract accuracy. https://github.com/tesseract-ocr/tesseract/wiki/4.0-Accuracy-and-Performance, .

[10] VirusTotal. https://www.virustotal.com/.

[11] Perceptual hash, 2020. https://www.phash.org/.

[12] Sadia Afroz and Rachel Greenstadt. Phishzoo: Detecting phishing websites by looking at them. In *Proc. of ICSC*, 2011.

[13] Pieter Agten, Wouter Joosen, Frank Piessens, and Nick Nikiforakis. Seven months' worth of mistakes: A longitudinal study of typosquatting abuse. In *Proc. of NDSS*, 2015.

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

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

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

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

[18] Apple. About safari international domain name support, 2016. https://support.apple.com/kb/TA22996?locale=en_US&viewlocale=en_US.

[19] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (GANs). In *Proc. of ICML*, 2017.

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

[21] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *CoRR*, abs/1802.00420, 2018.

[22] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *Proc. of ICML*, 2018.

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

[24] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Proc. of NIPS*, 2016.

[25] Steven Bird and Edward Loper. NLTK: the natural language toolkit. In *Proc. of ACL*, 2004.

[26] Aaron Blum, Brad Wardman, Thamar Solorio, and Gary Warner. Lexical feature based phishing url detection using online learning. In *Proc. of AISec*, 2010.

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

[28] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. Meerkat: Detecting website defacements through image-based object recognition. In *Proc. of USENIX Security*, 2015.

[29] Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger Gunn, Alexander Hammers, David Alexander Dickie, Maria Valdés Hernández, Joanna Wardlaw, and Daniel Rueckert. GAN augmentation: Augmenting training data using generative adversarial networks. *CoRR abs/1810.10863*, 2018.

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

[31] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *CoRR*, abs/1712.09665, 2017.

[32] Patricia Callejo, Rubén Cuevas, and Áangel Cuevas. An Ad-driven measurement technique for monitoring the browser marketplace. *IEEE Access*, 7, 2019.

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

[34] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Proc. of S&P (Okland)*, 2017.

[35] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *Proc. of IEEE S&P*, 2017.

[36] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *Proc. of ICCV*, 2017.

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

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

[39] Daiki Chiba, Ayako Akiyama Hasegawa, Takashi Koide, Yuta Sawabe, Shigeki Goto, and Mitsuaki Akiyama. Domainscouter: Understanding the risks of deceptive IDNs. In *Proc. of RAID*, 2019.

[40] Hyunsang Choi, Bin B Zhu, and Heejo Lee. Detecting malicious web links and identifying their attack types. In *Proc. of USENIX Conference on Web Application Development*, 2011.

[41] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proc. of CVPR*, 2017.

[42] Moustapha Cisse, Yossi Adi, Natalia Neverova, and Joseph Keshet. Houdini: Fooling deep structured visual and speech recognition models with adversarial examples. In *Proc. of NIPS*, 2017.

[43] Igino Corona, Battista Biggio, Matteo Contini, Luca Piras, Roberto Corda, Mauro Mereu, Guido Mureddu, Davide Ariu, and Fabio Roli. Deltaphish: Detecting phishing webpages in compromised websites. In *Proc. of ESORICS*, 2017.

[44] Adam Costello. Punycode: A bootstring encoding of unicode for internationalized domain names in applications (IDNA). RFC 3492, 2003. https://tools.ietf.org/html/rfc3492.

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

[46] D. Crocker, T. Hansen, and M. Kucherawy. Domainkeys identified mail (DKIM) signatures, 2011. https://tools.ietf.org/html/rfc6376.

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

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

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

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

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

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

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

[54] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proc. of CHI*, 2006.

[55] Guneet S. Dhillon, Kamyar Azizzadenesheli, Zachary C. Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. Stochastic activation pruning for robust adversarial defense. In *Proc. of ICLR Workshop*, 2018.

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

[57] Yueqi Duan, Jiwen Lu, and Jie Zhou. Uniformface: Learning deep equidistributed representation for face recognition. In *Proc. of CVPR*.

[58] Matthew Dunlop, Stephen Groat, and David Shelly. Goldphish: Using images for content-based phishing analysis. In *Proc. of ICIMP*, 2010.

[59] Manuel Egele, Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Towards detecting compromised accounts on social networks. In *Proc. of IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2017.

[60] Yahia Elsayed and Ahmed Shosha. Large scale detection of IDN domain name masquerading. In *Proc. of eCrime*, 2018.

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

[62] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. In *Proc. of CVPR*, 2018.

[63] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Dawn Song, Tadayoshi Kohno, Amir Rahmati, Atul Prakash, and Florian Tramèr. Note on attacking object detectors with adversarial stickers. *CoRR*, abs/1712.08062, 2017.

[64] Reuben Feinman, Ryan R. Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting Adversarial Samples from Artifacts. *CoRR*, abs/1703.00410, 2017.

[65] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. In *Proc. of WWW*, 2007.

[66] Volker Fischer, Mummadi Chaithanya Kumar, Jan Hendrik Metzen, and Thomas Brox. Adversarial examples for semantic image segmentation. *CoRR*, abs/1707.08945, 2017.

[67] Felicia Fonseca and Tim Krisher. Uber suspends self-driving car tests after pedestrian death in arizona. Chicago Tribune, 2018.

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

[69] David Mandell Freeman. Can you spot the fakes? On the limitations of user feedback in online social networks. In *Proc. of WWW*, 2017.

[70] Mattias Geniar. Show idn punycode in firefox to avoid phishing urls, 2018. https://ma.ttias.be/show-idn-punycode-firefox-avoid-phishing-urls/.

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

[72] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *Proc. of ICLR*, 2014.

[73] Google. Internationalized domain names (IDN) in google chrome, 2020. https://chromium.googlesource.com/chromium/src/+/master/docs/idn.md.

[74] Google. Tesseract orc, 2020. https://opensource.google/projects/tesseract.

[75] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *Proc. of ESORICS*, 2017.

[76] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proc. of NDSS*, 2008.

[77] Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. In *Proc. of ICLR*, 2014.

[78] Xiao Han, Nizar Kheir, and Davide Balzarotti. Phisheye: Live monitoring of sandboxed phishing kits. In *Proc. of CCS*, 2016.

[79] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[80] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defenses: Ensembles of weak defenses are not strong. In *Proc of WOOT*, 2017.

[81] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On Detecting Adversarial Perturbations. In *Proc. of ICLR*, 2017.

[82] Dan Hendrycks and Kevin Gimpel. Visible progress on adversarial images and a new saliency map. In *Proc. of ICLR Workshop*, 2017.

[83] Geoffrey Hinton, li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Phuongtrang Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 2012.

[84] Quan Hoang, Tu Dinh Nguyen, Trung Le, and Dinh Phung. MGAN: Training generative adversarial nets with multiple generators. In *Proc. of ICLR*, 2018.

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

[86] Tobias Holgers, David E. Watson, and Steven D. Gribble. Cutting through the confusion: A measurement study of homograph attacks. In *Proc. of USENIX ATC*, 2006.

[87] Tobias Holgers, David E Watson, and Steven D Gribble. Cutting through the confusion: A measurement study of homograph attacks. In *USENIX ATC*, 2006.

[88] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[89] Hang Hu and Gang Wang. End-to-end measurements of email spoofing attacks. In *Proc. of USENIX*, 2018.

[90] Hang Hu and Gang Wang. End-to-end measurements of email spoofing attacks. In *Proc. of USENIX Security*, 2018.

[91] Hang Hu, Peng Peng, , and Gang Wang. Towards understanding the adoption of anti-spoofing protocols in email systems. In *Proc. of SecDev*, 2018.

[92] Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. In *Proc. of ICLR Workshop*, 2017.

[93] IETF.org. Internationalizing domain names in applications (IDNA), 2003. https://tools.ietf.org/html/rfc3490.

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

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

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

[97] Steve T. K. Jan, Joseph Messou, Yen-Chen Lin, Jia-Bin Huang, and Gang Wang. Connecting the digital and physical world: Improving the robustness of adversarial attacks. In *Proc. of AAAI*, 2019.

[98] Steve T. K. Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, , and Bimal Viswanath. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *Proc. of IEEE S&P*, 2020.

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

[100] Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. Robust convolutional neural networks under adversarial noise. *CoRR*, abs/1511.06306, 2015.

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

[102] Kai Kang, Wanli Ouyang, Hongsheng Li, and Xiaogang Wang. Object detection from video tubelets with convolutional neural networks. In *CVPR*, 2016.

[103] Scott Kaplan, Benjamin Livshits, Benjamin Zorn, Christian Seifert, and Charles Curtsinger. "nofus: Automatically detecting" + string.fromcharcode(32) + "obfuscated ".tolowercase() + "javascript code". Technical Report MSR-TR-2011-57, Microsoft Research, May 2011.

[104] Mahmoud Khonji, Youssef Iraqi, and Andrew Jones. Phishing detection: a literature survey. *IEEE Communications Surveys & Tutorials*, 15(4):2091–2121, 2013.

[105] Jinkyu Kim, Teruhisa Misu, Yi-Ting Chen, Ashish Tawari, and John Canny. Grounding human-to-vehicle advice for self-driving vehicles. In *Proc. of CVPR*, 2019.

[106] Panagiotis Kintis, Najmeh Miramirkhani, Charles Lever, Yizheng Chen, Rosa Romero-Gómez, Nikolaos Pitropakis, Nick Nikiforakis, and Manos Antonakakis. Hiding in plain sight: A longitudinal study of combosquatting abuse. In *Proc. of CCS*, 2017.

[107] Scott Kitterman. Sender policy framework (SPF), 2014. https://tools.ietf.org/html/rfc7208.

[108] John Klensin and YangWoo Ko. Overview and framework for internationalized email. RFC4952, 2007. https://tools.ietf.org/html/rfc4952.

[109] Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies. In *Proc. of ICLR Workshop*, 2017.

[110] M. Kucherawy and E. Zwicky. Domain-based message authentication, reporting, and conformance (DMARC), 2015. https://tools.ietf.org/html/rfc7489.

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

[112] Ponnurangam Kumaraguru, Yong Rhee, Alessandro Acquisti, Lorrie Faith Cranor, Jason Hong, and Elizabeth Nunge. Protecting people from phishing: the design and evaluation of an embedded training email system. In *Proc. of CHI*, 2007.

[113] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Proc. of ICLR Workshop*, 2017.

[114] LambdaTest. Lambdatest: Cross browser testing cloud, 2020. https://www.lambdatest.com/.

[115] Victor Le Pochat, Tom Van Goethem, and Wouter Joosen. Funny accents: Exploring genuine interest in internationalized domain names. In *Proc. of PAM*, 2019.

[116] Eunjo Lee, Jiyoung Woo, Hyoungshick Kim, Aziz Mohaisen, and Huy Kang Kim. You are a game bot! Uncovering game bots in MMORPGs via self-similarity in the wild. In *Proc. of NDSS*, 2016.

[117] Xinzhe Li, Qianru Sun, Yaoyao Liu, Qin Zhou, Shibao Zheng, Tat-Seng Chua, and Bernt Schiele. Learning to self-train for semi-supervised few-shot classification. In *Proc. of NIPS*. 2019.

[118] Bin Liang, Miaoqiang Su, Wei You, Wenchang Shi, and Gang Yang. Cracking classifiers for evasion: A case study on the google's phishing pages filter. In *Proc. of WWW*, 2016.

[119] Eric Lin, Saul Greenberg, Eileah Trotter, David Ma, and John Aycock. Does domain highlighting help people identify phishing sites? In *Proc. of CHI*, 2011.

[120] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. In *Proc. of IJCAI*, 2017.

[121] Baojun Liu, Chaoyi Lu, Zhou Li, Ying Liu, Hai-Xin Duan, Shuang Hao, and Zaifeng Zhang. A reexamination of internationalized domain names: The good, the bad and the ugly. In *Proc. of DSN*, 2018.

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

[123] Jiajun Lu, Hussein Sibai, and Evan Fabry. Adversarial examples that fool detectors. *CoRR*, abs/1712.02494, 2017.

[124] Jiajun Lu, Hussein Sibai, Evan Fabry, and David A. Forsyth. NO Need to Worry about Adversarial Examples in Object Detection in Autonomous Vehicles. In *CVPR Workshop*, 2017.

[125] Meng Luo, Oleksii Starov, Nima Honarmand, and Nick Nikiforakis. Hindsight: Understanding the evolution of ui vulnerabilities in mobile browsers. In *Proc. of CCS*, 2017.

[126] Meng Luo, Pierre Laperdrix, Nima Honarmand, and Nick Nikiforakis. Time does not heal all wounds: A longitudinal analysis of security-mechanism support in mobile browsers. In *Proc. of NDSS*, 2019.

[127] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Learning to detect malicious urls. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2011.

[128] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *Proc. of ICLR Workshop*, 2017.

[129] Larry M. Manevitz and Malik Yousef. One-class SVMs for document classification. *Journal of Machine Learning Research*, 2002.

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

[131] Samuel Marchal, Kalle Saari, Nidhi Singh, and N Asokan. Know your phish: Novel techniques for detecting phishing sites and their targets. In *Proc. of ICDCS*, 2016.

[132] Eric Medvet, Engin Kirda, and Christopher Kruegel. Visual-similarity-based phishing detection. In *Proc. of SecureComm*, 2008.

[133] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proc of CCS*, 2017.

[134] Microsoft. Changes to IDN in IE7 to now allow mixing of scripts, 2006. https://docs.microsoft.com/en-us/archive/blogs/ie/changes-to-idn-in-ie7-to.

[135] Microsoft. Lifecycle FAQ - Internet explorer and Edge, 2016. https://docs.microsoft.com/en-us/lifecycle/faq/internet-explorer-microsoft-edge.

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

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

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

[139] Paul Mockapetris. Domain names - concepts and facilities. RFC 1034, 1987. https://tools.ietf.org/html/rfc1034.

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

[141] Tyler Moore and Benjamin Edelman. Measuring the perpetrators and funders of typosquatting. In *International Conference on Financial Cryptography and Data Security*, 2010.

[142] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *Proc. of CVPR*, 2016.

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

[144] Mozilla. Firefox IDN display algorithm, 2017. https://wiki.mozilla.org/IDN_Display_Algorithm.

[145] NetMarketShare. Browser market share, 2020. https://netmarketshare.com/browser-market-share.aspx.

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

[147] Nick Nikiforakis, Steven Van Acker, Wannes Meert, Lieven Desmet, Frank Piessens, and Wouter Joosen. Bitsquatting: Exploiting bit-flips for fun, or profit? In *Proc. of WWW*, 2013.

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

[149] Adam Oest, Yeganeh Safei, Adam Doupe, Gail-Joon Ahn, Brad Wardman, and Gary Warner. Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis. In *Proc. of eCrime*, 2018.

[150] U.S. Department oF Homeland Security. U.s. department of homeland security cybersecurity strategy, 2018. URL https://www.dhs.gov/sites/default/files/publications/DHS-Cybersecurity-Strategy_1.pdf.

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

[152] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *Proc. of IEEE Euro. S&P*, 2016.

[153] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proc. ASIA CCS*, 2017.

[154] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. In *Proc. of AsiaCCS*, 2017.

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

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

[157] Sai Prashanth Chandramouli, Pierre-Marie Bajan, Christopher Kruegel, Giovanni Vigna, Ziming Zhao, Adam Doup, and Gail-Joon Ahn. Measuring e-mail header injections on the world wide web. In *Proc. of SAC*, 2018.

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

[159] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *Proc. of ICLR*, 2018.

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

[161] Bradley Reaves, Nolen Scaife, Dave Tian, Logan Blue, Patrick Traynor, and Kevin RB Butler. Sending out an sms: Characterizing the security of the sms ecosystem with public gateways. In *Proc. of S&P (Okland)*, 2016.

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

[163] P. Resnick and P. Hoffman. Mapping characters for internationalized domain names in applications (IDNA). RFC 5895, 2008. https://tools.ietf.org/html/rfc5895.

[164] Joshua Reynolds, Deepak Kumar, Zane Ma, Rohan Subramanian, Meishan Wu, Martin Shelton, Joshua Mason, Emily Stark, and Michael Bailey. Measuring identity confusion with uniform resource locators. In *Proc. of CHI*, 2020.

[165] Alexei A. Efros Eli Shechtman Oliver Wang. Richard Zhang, Phillip Isola. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. of CVPR*, 2018.

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

[167] Andras Rozsa, Ethan M. Rudd, and Terrance E. Boult. Adversarial diversity and hard positive generation. In *Proc. of CVPR Workshop*, 2016.

[168] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

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

[170] Stuart Schechter, Rachna Dhamija, Andy Ozment, and Ian C Fischer. The emperor's new security indicators an evaluation of website authentication and the effect of role playing on usability studies. In *Proc. of IEEE SP*, 2007.

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

[172] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proc. of CCS*, 2016.

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

[174] Chawin Sitawarin, Arjun Nitin Bhagoji, Arsalan Mosenia, Mung Chiang, and Prateek Mittal. DARTS: deceiving autonomous cars with toxic signs. *CoRR*, abs/1802.06430, 2018.

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

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

[177] Congzheng Song and Vitaly Shmatikov. Fooling OCR systems with adversarial text images. *CoRR*, abs/1802.05385, 2018.

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

[179] StatCounter. Browser market share worldwide, 2020. https://gs.statcounter.com/browser-market-share.

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

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

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

[183] Hiroaki Suzuki, Daiki Chiba, Yoshiro Yoneya, Tatsuya Mori, and Shigeki Goto. Shamfinder: An automated framework for detecting IDN homographs. In *Proc. of IMC*, 2019.

[184] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proc. of ICLR*, 2014.

[185] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proc. of CVPR*, 2016.

[186] Janos Szurdi, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, and Chris Kanich. The long" taile" of typosquatting domain names. In *Proc. of USENIX Security*, 2014.

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

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

[189] Hao Tang, Dan Xu, Wei Wang, Yan Yan, and Nicu Sebe. Dual generator generative adversarial networks for multi-domain image-to-image translation. In *Proc. of ACCV*, 2018.

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

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

[192] Christopher Thompson, Martin Shelton, Emily Stark, Maximilian Walker, Emily Schechter, and Adrienne Porter Felt. The web's identity crisis: Understanding the effectiveness of website identity indicators. In *Proc. of USENIX Security*, 2019.

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

[194] Ke Tian, Zhou Li, Kevin Bowers, and Danfeng Yao. FrameHanger: Evaluating and classifying iframe injection at large scale. In *Proc. of SecureComm*, 2018.

[195] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[196] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble Adversarial Training: Attacks and Defenses. *CoRR*, abs/1705.07204, 2017.

[197] Unicode.org. Unicode confusables, 2015. https://www.unicode.org/Public/security/8.0.0/confusables.txt.

[198] Unicode.org. Unicode 13.0.0, 2020. https://unicode.org/versions/Unicode13.0.0/.

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

[200] W3Counter. Browser & platform market share, 2020. https://www.w3counter.com/globalstats.php.

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

[202] Gang Wang, Tianyi Wang, Haitao Zheng, and Ben Y Zhao. Man vs. Machine: Practical adversarial detection of malicious crowdsourcing workers. In *Proc. of USENIX Security*, 2014.

[203] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G. Ororbia, II, Xinyu Xing, Xue Liu, and C. Lee Giles. Adversary resistant deep neural networks with an application to malware detection. In *Proc. of KDD*, 2017.

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

[205] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proc. of CVPR*, 2018.

[206] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE TIP*, 13(4), 2004.

[207] Liu Wenyin, Guanglin Huang, Liu Xiaoyue, Zhang Min, and Xiaotie Deng. Detection of phishing webpages based on visual similarity. In *Proc. of WWW*, 2005.

[208] Guang Xiang, Jason Hong, Carolyn P Rose, and Lorrie Cranor. Cantina+: A feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security (TISSEC)*, 2011.

[209] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *Proc. of ICCV*, 2017.

[210] Wei Xu, Fangfang Zhang, and Sencun Zhu. Jstill: mostly static detection of obfuscated malicious javascript code. In *Proc. of AsiaCCS*, 2013.

[211] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proc. of NDSS*, 2018.

[212] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *Proc. of NDSS*, 2018.

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

[214] Zili Yi, Hao Zhang, Ping Tan, and Minglun Gong. DualGAN: Unsupervised dual learning for image-to-image translation. In *Proc. of ICCV*, 2017.

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

[216] Yue Zhang, Jason I Hong, and Lorrie F Cranor. Cantina: a content-based approach to detecting phishing web sites. In *Proc. of WWW*, 2007.

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

[218] Haizhong Zheng, Minhui Xue, Hao Lu, Shuang Hao, Haojin Zhu, Xiaohui Liang, and Keith W. Ross. Smoke screener or straight shooter: Detecting elite sybil attacks in user-review social networks. In *Proc. of NDSS*, 2018.

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

[220] Stephan Zheng, Yang Song, Thomas Leung, and Ian J. Goodfellow. Improving the robustness of deep neural networks via stability training. In *Proc. of CVPR*, 2016.

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

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

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

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

Appendix

Appendix A: LSTM vs. CNN

To justify our choice of Long-Short-Term-Memory (LSTM) model [204], 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 6.1, 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 6.1: 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 6.2. 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 6.2.

Table 6.2: 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 6.3: Feeding synthetic data to LSTM and RF. We use August-18 dataset from Website B; Models are trained with 1% of the training dataset.

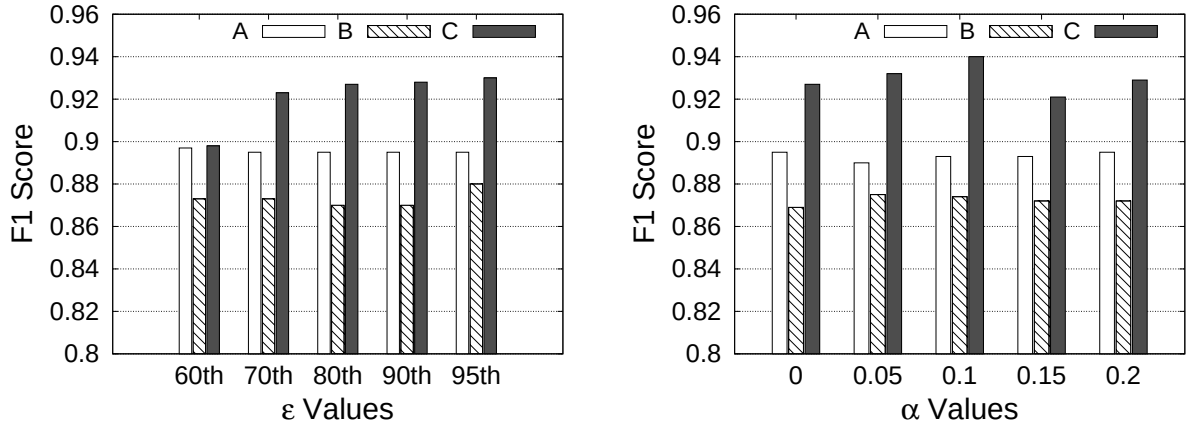
	Synthetic data?	Precision	Recall	F1
RF	No	0.883	0.202	0.343
	Yes	0.826	0.570	0.596
LSTM	No	0.601	0.355	0.446
	Yes	0.698	0.757	0.719
ODDS	Yes	0.729	0.845	0.783

As shown in Table 6.3, 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 dissertation, 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 6.1a 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 6.1b 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.



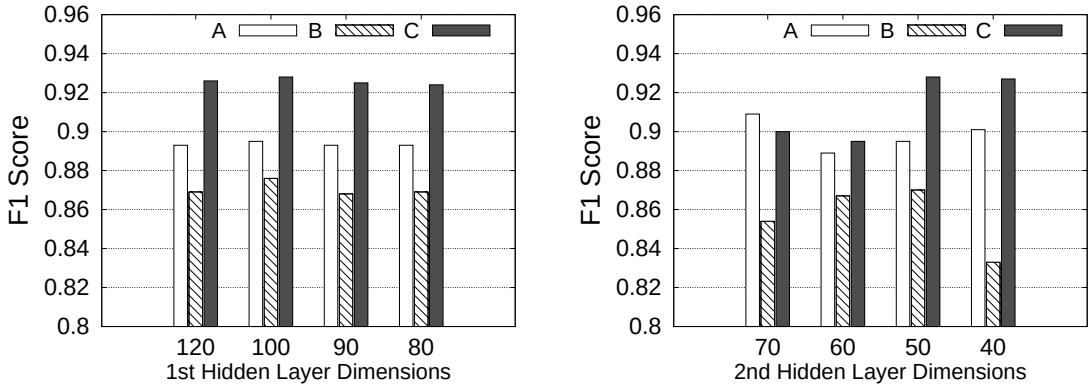
(a) F1 score for different ϵ

(b) F1 score for different α

Figure 6.1: F1 score on the *validation set* for different ϵ and α for website A, B, C.

Figure 6.2 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.

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 4.5). 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.



(a) F1 score for different dimensions of first layer

(b) F1 score for different dimensions of second layer

Figure 6.2: F1 scores on the *validation set* using different dimensions for layers in the generators and the discriminator for website A, B, C.

Table 6.4 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.

Appendix F: False Positives and False Negatives

To complement the main results in Table 4.8, we add a new Table 6.5 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 4.8, and Table 6.5, we show that our system ODDS can drastically increase the number of detected true bots (reducing false negative rate) while producing comparable number of

Table 6.4: 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

Table 6.5: 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

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).

Appendix G: User Study Questions

Demographic question 1: What is your gender?

- Male
- Female
- Others (please specify)

Demographic question 2: How old are you?

- 18-29
- 30-39
- 40-49
- 50 or above
- Prefer not to answer

Demographic question 3: What's your highest education/degree completed?

- High school graduate or less
- Some college or two-year associate degree
- Bachelor's degree

- Some graduate school
- Master's or professional degree
- Ph.D.
- Prefer not to answer

Demographic question 4: How long have you been using web browsers?

- Less than a year
- 1-3 years
- 3-5 years
- More than 5 years
- Prefer not to answer

Demographic question 5: Do you have a technical background in computing?

- Yes
- No
- Prefer not to answer

Internet usage question: On average, how often do you visit these different categories of websites? For each of the 5 categories: shopping websites, banking websites, social networking websites, eduaction websites and government and military websites, participants can answer one of the following 5 options.

- Multiple times a day
- Once a day
- Once a week
- Once a month
- Once a year
- Less than once a year

Appendix J: IDN in Email Clients

The protocol to support IDN in email address is called Email Address Internationalization (EAI) [108]. With EAI, email clients can display IDNs in Unicode when they are used as the domain name of email addresses. As such, attackers can run IDN homograph to impersonate trusted email senders. More specifically, the attacker can register the IDN (*e.g.*, the Cyrillic "apple.com") that looks like target domain names (the real apple.com) for spear phishing. Compared to email spoofing [90], IDN based impersonation is harder to detect. This is

Email Clients	**IDN Support**	**Homograph Attack**
Gmail (Web)	Yes	Succeeded
iCloud (Web)	No	Failed
Outlook (Web)	No	Failed
Gmail (Mobile)	Yes	Succeeded
iCloud (Mobile)	Yes	Succeeded
Outlook (Mobile)	No	Failed
Messaging Apps	**IDN Support**	**Homograph Attack**
Facebook Post	No	Failed
Twitter Post	Yes	Succeeded
Messenger	Yes	Succeeded
iMessage	Yes	Succeeded
Whatsapp	Yes	Succeeded

Table 6.6: IDN policies in email services and messaging apps.

because the attacker actually owns the Cyrillic apple.com, and thus the attacker does not need to run any "spoofing." As a result, anti-spoofing protocols such as SPF [107], DKIM [46] and DMARC [110] cannot block the attack.

Testing IDN homograph attacks on email systems is more difficult, since we need to bypass the spam check and potentially anti-spoofing protocols (*e.g.*, SPF, DKIM, DMARC) in order to display the emails. This requires us to purchase the testing IDN domains and set up the proper DNS records.

Testing Methodology and Results In May 2020, we registered 3 IDNs. The first IDN is " .com". This IDN does not impersonate any target domain name. It represents legitimate usage of internationalized domain names and does not violate any known IDN policies. We use this IDN to test if the email clients support IDN. The second IDN is "googīe.com". This IDN impersonates "google.com". This IDN represents homograph IDNs that can bypass browser defense. The third IDN is "þaidu.com". This IDN impersonates "baidu.com". This IDN represents homograph IDNs that can be detected by browser policies (*e.g.*, by Chrome). We use those IDNs as email domain names and send emails to popular email services. We then examine if the sender email address will be displayed as Unicode.

For each IDN, we first set up its DNS record, as well as the SPF and DKIM records so that they can pass anti-spoofing protocols. Then we crafted testing emails and sent them to *our own accounts* created in three popular public email providers: Gmail, iCloud, and Outlook. By using trust-worthy IPs to send emails, we successfully pushed the emails to the inboxes of receivers (our own accounts). Then we displayed the emails on the web and mobile interfaces, and checked whether the email clients show Unicode.

As shown in Table 6.6, at least Gmail web and mobile clients and iCloud mobile clients supported IDN, and they displayed " .com" in Unicode. Others did not show Unicode for IDN by default. Then for the two homograph IDNs, if an email client displays at least

Figure 6.3: "googīe.com" in Gmail web interface.

one homograph IDN as Unicode, we regard the homograph attack is successful.

As shown in Table 6.6, for all service clients that support IDN, we succeed in conducting a homograph attack. We found that both Gmail and iCloud were displaying homograph IDN in Unicode in the email address field. A screenshot for Gmail web client is shown in Figure 6.3. In fact, both baidu.com and googīe.com are successfully displayed in both Gmail web and mobile email clients. Interestingly, the IDN homograph attack succeeds in iCloud only on the mobile client, but not the web clients. For Outlook, we found that the email services did not display Unicode for IDN by default. Overall, we find homograph IDNs work on all the tested email clients as long as they support IDN. The current email services have not yet adopted the same level of defense as web browsers.

As a side note, we also tried to put the testing IDN as part of the URL in the email content in the *email body*. We found all four email providers display the Unicode in the email body.

Appendix K: IDN in Social Messaging Apps

We have examined how social messaging apps display IDNS in the message content. Social messaging apps can be used to disseminate phishing links composed with IDNs. To test their IDN policies (if any), we randomly select 100 target domain names from Alexa top 1 million domains, and created a homograph IDN to impersonate the target domain name.

With the 100 testing IDNs, we conducted a test in May 2020 on Facebook, Twitter, Messenger, iMessage, and Whatsapp. For Facebook and Twitter, we put the links into a post (or several posts) to examine whether the link is displayed as Unicode. After the test, we immediately delete the posts and the testing accounts. For messaging apps, we put the links into messages and send the messages using these apps to *our own accounts*.

We found that only Facebook would convert *all the testing IDNs* into the Punycode form. Twitter, Messenger, and iMessage would display all testing cases in the Unicode form. The results indicate that most social messaging apps are not yet established the defense against IDN homograph yet (except for Facebook posts).