

# D-FENS: DNS Filtering & Extraction Network System for Malicious Domain Names

2018

Jeffrey Spaulding  
University of Central Florida

Find similar works at: <https://stars.library.ucf.edu/etd>

University of Central Florida Libraries <http://library.ucf.edu>

 Part of the [Computer Sciences Commons](#)

## STARS Citation

Spaulding, Jeffrey, "D-FENS: DNS Filtering & Extraction Network System for Malicious Domain Names" (2018). *Electronic Theses and Dissertations*. 6378.

<https://stars.library.ucf.edu/etd/6378>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact [lee.dotson@ucf.edu](mailto:lee.dotson@ucf.edu).

D-FENS: DNS FILTERING & EXTRACTION NETWORK SYSTEM FOR MALICIOUS  
DOMAIN NAMES

by

JEFFREY SPAULDING  
B.S. Clarkson University, 2003  
M.S. SUNY Polytechnic Institute, 2013

A dissertation submitted in partial fulfilment of the requirements  
for the degree of Doctor of Philosophy  
in the Department of Computer Science  
in the College of Engineering and Computer Science  
at the University of Central Florida  
Orlando, Florida

Summer Term  
2018

Major Professor: Aziz Mohaisen

© 2018 Jeffrey Spaulding

## ABSTRACT

While the DNS (Domain Name System) has become a cornerstone for the operation of the Internet, it has also fostered creative cases of maliciousness, including phishing, typosquatting, and botnet communication among others. To address this problem, this dissertation focuses on identifying and mitigating such malicious domain names through prior knowledge and machine learning. In the first part of this dissertation, we explore a method of registering domain names with deliberate typographical mistakes (i.e., typosquatting) to masquerade as popular and well-established domain names. To understand the effectiveness of typosquatting, we conducted a user study which helped shed light on which techniques were more successful than others in deceiving users. While certain techniques fared better than others, they failed to take the context of the user into account. Therefore, in the second part of this dissertation we look at the possibility of an advanced attack which takes context into account when generating domain names. The main idea is determining the possibility for an adversary to improve their success rate of deceiving users with specifically-targeted malicious domain names. While these malicious domains typically target users, other types of domain names are generated by botnets for command & control (C2) communication. Therefore, in the third part of this dissertation we investigate domain generation algorithms (DGA) used by botnets and propose a method to identify DGA-based domain names. By analyzing DNS traffic for certain patterns of NXDomain (non-existent domain) query responses, we can accurately predict DGA-based domain names before they are registered. Given all of these approaches to malicious domain names, we ultimately propose a system called D-FENS (DNS Filtering & Extraction Network System). D-FENS uses machine learning and prior knowledge to accurately predict unreported malicious domain names in real-time, thereby preventing Internet devices from unknowingly connecting to a potentially malicious domain name.

This dissertation is dedicated to my wife Nicole, my sons Greyson and Landon, and to my dear parents who all encouraged me to accomplish this wonderful journey.

## **ACKNOWLEDGMENTS**

I would like to express my sincere gratitude to my advisor and doctoral committee chair, Dr. Aziz Mohaisen, who saw potential in me early on and shaped me for success in researching and publishing scholarly work. I would also like to thank the members of my committee for their valuable feedback and support: Dr. Gary Leavens, Dr. Mostafa Bassiouni, Dr. Xinwen Fu, and Dr. Clay Posey. Last but not least, I want to thank my teammates in the Security Analytics Lab (SEAL) for their encouragement and support.

## TABLE OF CONTENTS

LIST OF FIGURES . . . . .	ix
LIST OF TABLES . . . . .	xi
CHAPTER 1: INTRODUCTION . . . . .	1
1.1 Statement of Research . . . . .	2
1.2 Approach . . . . .	3
1.3 Contributions . . . . .	5
1.4 Dissertation Organization . . . . .	6
CHAPTER 2: THE EFFECTIVENESS OF MALICIOUS & TYPOSQUATTED DOMAIN NAMES . . . . .	8
2.1 Motivation . . . . .	8
2.2 Background & Related Work . . . . .	11
2.3 User Study: Identifying Typo Domains . . . . .	18
2.4 Design of the User Study . . . . .	21
2.5 Results & Discussion . . . . .	23
2.6 Conclusions & Recommendations . . . . .	31

CHAPTER 3: MALICIOUS & TYPOSQUATTED DOMAIN NAME ATTACKS WITH CONTEXT . . . . .	33
3.1 Background & Related Work . . . . .	33
3.2 User Study: Malicious Domain Names with Context . . . . .	39
3.3 User Study Results . . . . .	43
3.4 Conclusion . . . . .	47
CHAPTER 4: PROACTIVE DETECTION OF ALGORITHMICALLY GENERATED DO- MAIN NAMES . . . . .	49
4.1 Motivation . . . . .	51
4.2 Related Work . . . . .	52
4.3 Preliminaries . . . . .	55
4.4 Dataset and Characteristics . . . . .	57
4.5 Online Detection Algorithm . . . . .	61
4.6 Threat Model and System Overview . . . . .	68
4.7 Evaluation . . . . .	70
4.8 Discussion . . . . .	72
4.9 Conclusion . . . . .	77



CHAPTER 5: DNS FILTERING & EXTRACTION NETWORK SYSTEM . . . . .	78
5.1 Background & Related Work . . . . .	79
5.2 D-FENS System Model . . . . .	82
5.3 Evaluation . . . . .	87
5.4 Summary & Future Work . . . . .	90
CHAPTER 6: CONCLUSION . . . . .	92
APPENDIX : COPYRIGHT INFORMATION . . . . .	94
REFERENCES . . . . .	102

## LIST OF FIGURES

Figure 2.1: Phase 1 of the malicious domain name user study. . . . .	21
Figure 2.2: Screenshots of phases 2 & 3 of the malicious domain name user study. . . . .	22
Figure 2.3: Plots of the study results depicting correct responses and completion time per phase. . . . .	24
Figure 2.4: Plots of the user study results according to age. . . . .	25
Figure 2.5: Plots of the user study results according to education and familiarity . . . . .	26
Figure 2.6: Plots of the user study results according to domain ranking and typo model. . . . .	27
Figure 2.7: Plots of the user study results according to domain type and TLD. . . . .	29
Figure 3.1: An illustration of the steps of DNS resolution through the recursive, root, TLD, and authoritative name server. . . . .	35
Figure 3.2: An example of a browser extension using the <i>chrome.history</i> API to extract the browser's history. . . . .	38
Figure 3.3: Screenshots of screens 1 & 2 of the context user study. . . . .	40
Figure 3.4: User-supplied domains from the context user study. . . . .	42
Figure 3.5: JavaScript code snippet for generating a typosquatted domain name. . . . .	43

Figure 3.6: Overall correct responses and completion time, as well as correct responses by age. . . . .	44
Figure 3.7: Plots of the context user study according to education and ethnicity. . . . .	45
Figure 3.8: Plots of the context user study according to correct/incorrect by typo model. . . . .	46
Figure 4.1: An illustration of a botnet infrastructure. . . . .	49
Figure 4.2: NXDomain traffic volumes to major TLDs: .cc, .tv, .com, and .net, respectively. . . . .	59
Figure 4.3: Conficker NXDomain DNS lookups. . . . .	60
Figure 4.4: An example of the algorithm used by CryptoLocker for initially generating algorithmic domain names for command and control . . . . .	62
Figure 4.5: A system diagram for the detection of DGA-based malicious domains. . . . .	69
Figure 4.6: Plotting the standard measurements of performance: false negative/true positive, false positive/true negative for different windows size (average, over 24 slides for the given $W$ size). Notice that an accuracy of 0.99 can be achieved for more than 48 hours in advance. . . . .	72
Figure 5.1: An overview of the D-FENS architecture. . . . .	83
Figure 5.2: ROC curves for each fold of the cross validation of the deep learning model. . . . .	89
Figure 5.3: Sample prediction times for DNS queries showing elapsed times from 2.5 to 3.1 milliseconds. . . . .	90

## LIST OF TABLES

Table 2.1: Summary of Typo Domain Identification Approaches. . . . .	13
Table 2.2: Number of candidate domains per model . . . . .	20
Table 2.3: Top 10 incorrectly identified domains (unmodified domains shown in gray). . .	30
Table 2.4: Top 10 correctly identified domains (unmodified domains shown in gray). . .	31
Table 3.1: Top incorrectly identified user-supplied domains (unmodified shown in gray). .	46
Table 4.1: Summary of DGA Domain Identification Approaches. . . . .	53
Table 4.2: Conficker DGA by variant and domains per day. . . . .	58
Table 4.3: Standard measurements of performance: true positive, true negative, false positive, false negative for different windows size (average, over 24 slides for the given $W$ size). . . . .	71
Table 5.1: The data sets used in model training. . . . .	88

## CHAPTER 1: INTRODUCTION

In today's connected world, billions of devices are able to access the Internet to allow their users to connect and exchange information. At the heart of this global information infrastructure is the DNS, which allows people to use text-based human-readable domain names instead of numeric IP addresses to provide a distributed directory service. While the majority of all domain names ultimately resolve to a web server that hosts meaningful content, there are an alarming amount of malicious ones that involve nefarious activities such as phishing, typosquatting and botnet command & control (C2) channels among others.

Typosquatting, for example, are domain names that use deliberate typographical mistakes to imitate popular and well-known domain names. Typically, these typosquatted domains are used for malicious purposes such as redirecting web browsers to phishing sites or stealing web traffic away from the authoritative domains (for financial gain). Typosquatting has long been around for over twenty years and is still being used today. As recently uncovered on the popular cybercrime blog *KrebsOnSecurity.com* [70], there is “a vast network of potentially malicious Web sites ending in ‘.cm’ that mimic some of the world's most popular Internet destinations (e.g. `espn.cm`, `aol.cm` and `itunes.cm`)” which demonstrates a common typosquatting tactic: omitting a character from the top-level domain (TLD).

Additionally, domain names and the DNS infrastructure are increasingly being utilized by botnets to establish communication between bots and their C2 servers. The typical botnet consists of various infected hosts, C2 channels, and a botmaster. The infected hosts (“zombies”) are often massively distributed, whereas the command and control are channels used by a mastermind (the “botmaster”) to instruct bots to perform various forms of malice; e.g. launching distributed denial-of-service (DDoS) attacks [114]. To communicate with bots, there are several potential

ways utilized by botmasters, and domain names as a C2 channel are one of the most common and preferred methods—because they are easy to acquire and recycle [69]. One way of mitigating these botnets is to prevent them from registering their C2 domains in the first place, or by taking such domain names down [89, 77]. However, the adoption of domain generation algorithms (or DGAs), which can produce thousands of pseudo-random domain names, has made traditional domain takedowns and detection more difficult for law enforcement and security researchers [4].

## 1.1 Statement of Research

Whether it was from mistyping a URL into a web browser or clicking on a hyperlink with a deceptively-familiar domain name, malicious domain names which use intentional typos to appear similar as authoritative domains have long been pervasive in the DNS infrastructure since the late 1990's [51]. This so-called typosquatting attack involves applying different techniques such as adding or substituting characters to already established and well-known domain names in the hopes that it will deceive unsuspecting users in navigating to unwanted destinations. While previous work in the literature has identified all of the techniques that typosquatters employ in the wild [117, 31], there is little work that investigated the efficacy of such techniques and how successful they were at deceiving users. The first problem we would like to solve is how effective is typosquatting against the average Internet user and what techniques actually deceive them?

Building upon the first problem, a second question arises: If an adversary could take the context of a user into account when creating targeted malicious domain names, will their chances of successfully deceiving the user improve? The typical modus operandi of adversaries that engage in typosquatting will seek the best return on their investment and target the most-popular domain names to lure the most victims. Typosquatting a domain name that is well-known for a certain demographic of users may not have the same effect for other groups of users. For example,

tistory.com is a popular South Korean blog-publishing service with 91.0% of visitors originating from South Korea while only 1.6% are from the United States<sup>1</sup>. As such, the typical Internet user from the United States may not be familiar with this domain name and any attempts to deceive them with a typosquatted variant may fail.

The third issue we would like to explore is the fact that not all malicious domain names are crafted manually from human adversaries, but rather dynamically from domain generation algorithms (DGAs). As mentioned above, these generated domain names are employed by botnets to serve as rallying points for C2 servers to instruct the infected “zombie” machines to perform various cyber attacks, such as DDoS. Knowing that some infected machines will make out-of-sync DNS queries to the generated C2 domain names prior to them actually being registered (due to misconfiguration), is it possible to identify such DGA-based domain names before they are registered by the botmaster?

To address these issues, we seek to design experiments and create systems that will help us understand and identify malicious domain names. The overarching goal behind this research is to design and implement a tool that will identify and intercept malicious domain names in real-time, thereby preventing users and devices from unwanted destinations.

## 1.2 Approach

To tackle these issues, we conducted two user studies that resulted in a better understanding of which malicious domain names were more effective than others when used in typosquatting attacks. We also designed and implemented a system to proactively classify DGA-based domain names before they are registered with high accuracy. The results and methodologies of these ap-

---

<sup>1</sup>According to: <https://www.alexa.com/siteinfo/tistory.com>

proaches allowed us to develop a system that was capable of identifying malicious domain names in real-time.

In the first user study, we attempted to measure the effectiveness of several typosquatting techniques identified in the literature. We conducted a multi-phase user study that exposed participants to several URLs in which a subset were deliberately modified using known typosquatting techniques. Our results showed that the participant's correct identification of typosquatting improved (with quicker response times) after being incrementally shown all typo techniques. We also discovered which typosquatting models were more successful than others in deceiving users, and attributed it to studies in Cognitive Science that highlighted how the human brain encodes jumbled characters in words.

The second study explored the possibility of an Adversary taking a user's context into account when devising an attack strategy involving malicious domain names. The results of the first user study showed that while some typosquatting techniques fared better than others in deceiving users, the actual selection of the target domain names to typosquat may play a role. To this end, we conducted another online user study to establish context by dynamically fetching domain names based on user input (*e.g.* country, subject interest, favorite websites) and similarly applied a random typosquatting model. Our results confirmed which typosquatting strategies were the most successful for an Adversary when taking context into account.

For the third problem, we designed a system called DRIFT to proactively classify DGA-based malicious domain names before they are registered in the DNS by botmasters in an effort to sever their C2 communication. While previous efforts to identify and prevent DGA-domain names from operating have relied on reverse-engineering and intrinsic analysis, DRIFT uses DNS query analysis to look for tell-tale patterns of NXDomain responses. Ultimately, we achieved a 99% accuracy on average as early as 48 hours prior to the registration of a malicious DGA-based domain name.



For our final problem, we build upon these previous approaches and ultimately developed a system called D-FENS to detect and intercept malicious domain names in real-time. While typical defense systems employ the use of blacklists, such as *Google Safe Browsing*, we augment the use of blacklists by dynamically determining if a given domain name is malicious through prior knowledge and machine learning. Since blacklists offer an efficient means to lookup potentially-malicious domain names, they are only effective if such domain names are actually reported and added to the blacklist. As a result, there is a window of vulnerability for users to be exposed to malicious domain names allowing adversaries to wreak havoc until their domain names are blocked or taken down. As with the case of the aforementioned malicious activities involving both typosquatting and DGA-based domains used by malware, *domain names are low-cost and disposable resources*: once the adversary is done with the malicious activity, the domain names are most likely blocked or de-registered (within the grace period of registration; often up to 7 days allowed by registries). Ideally, D-FENS will act as a middlebox and intercept DNS requests before they enter the DNS resolution phase thereby preventing clients from ultimately connecting to a malicious IP address.

### 1.3 Contributions

In this dissertation we:

- Completed a comprehensive study on the landscape of the malicious domain name type known as typosquatting.
- Designed a user study for gauging how effective typosquatting is being used “in the wild”.
- Explored the possibility of an Adversary to take a users context into account when devising an attack strategy involving malicious domain names.

- Investigated a large-scale analysis of domain names and their NXDomain queries to understand and identify DGA-based domains.
- Designed a system capable of identifying and intercepting malicious domain names in real-time using machine learning.

In order to realize these contributions, we:

- Conducted an online user study in which participants were exposed to several URLs, a subset of which were deliberately typosquatted to determine their effectiveness. Our results showed that participant scores improved overall after being incrementally shown all typo techniques.
- Deployed a custom-developed online user study to establish context by dynamically fetching domain names based on user input and similarly applied a random typosquatting model. Our results found which typosquatting strategies were the most successful for an Adversary when taking context into account.
- Developed a simple classification algorithm for the proactive detection of DGA-based malicious domain names based on the NXDomain query patterns used by botnets. Our evaluation results showed high accuracy scores as early as 2 days prior to the registration of a DGA-based domain name.
- Implemented a new system that employs deep learning to produce a model that is used in a live DNS server to accurately predict malicious domain names in DNS queries in real-time.

#### 1.4 Dissertation Organization

This dissertation encompasses material from three papers by the author [105, 106, 107]. Chapter 2 uses material from Reference [105], coauthored with DaeHun Nyang and Aziz Mohaisen, which

presents the results of the user study to understand the effectiveness of malicious and typosquatted domain names. Chapter 3 explores the possibility of taking context into account when devising an attack strategy involving malicious & typosquatted domain names. Chapter 4 is based on Reference [106], coauthored with Jeman Park, Joongheon Kim, and Aziz Mohaisen, which presents a system called DRIFT to proactively detect algorithmically-generated malicious domain names typically employed by botnets. Finally, Chapter 5 proposes a system called D-FENS which will serve as the first line of defense against malicious domain names. Some material from each of these papers has also been incorporated into this introductory Chapter.

## CHAPTER 2: THE EFFECTIVENESS OF MALICIOUS & TYPOSQUATTED DOMAIN NAMES<sup>1</sup>

In this chapter, we describe the form of a malicious domain name that involves deliberately registering domains containing typographical errors so they appear similar to other (more popular) domain names. Typosquatting, as this practice became known as, exploits common typographical errors made by users that manually type URLs into web browsers in an attempt to steal traffic or redirect users to unintended destinations. While much of the existing literature has examined various typosquatting techniques and how they change over time, none have considered how effective they are in deceiving users. In this work, we attempt to fill in this gap by conducting a user study that exposes subjects to several uniform resource locators (URLs) in an attempt to determine the effectiveness of several typosquatting techniques that are prevalent in the wild. We also attempt to determine if the security education and awareness of cybercrimes such as typosquatting will affect the behavior of Internet users.

### 2.1 Motivation

Typosquatters employ several techniques in the wild to register domain names that can sufficiently capture enough traffic for monetary gain. For example, a typosquatter may target a popular domain name and register a typo variant in which only a single character is added or substituted. This is demonstrated by the famous case outlined in [97] where typosquatters targeted the immensely popular social-networking site *Facebook* to produce domain names such as `www.fagebook.com`

---

<sup>1</sup>This content was reproduced from the following article: J. Spaulding, D. Nyang, and A. Mohaisen, “Understanding the effectiveness of typosquatting techniques,” *In Proceedings of the Fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, HotWeb 17, pages 9:19:8, New York, NY, USA, 2017. ACM. The copyright form for this article is included in the appendix.

and `www.facebook.com`

**Approaches to Typosquatting.** Much of the research in identifying and understanding typosquatting generally falls under two approaches:

- (i) Identifying typosquatted domain names from domain name registration information.
- (ii) Identifying typosquatted domain names through network traffic analysis.

One of the first large-scale studies in the first approach was conducted in 2003 by Edelman [48], who located more than 8,800 registered domains that were minor typographical variations of popular domain names. A few years later, Wang *et al.* [117] introduced a system called the *Strider Typo-Patrol* for systematically identifying typo domains via “typo-generation models”. Subsequent studies including Banerjee *et al.* [31] and Edelman [49] utilized the typo-generation models in [117] to produce a corpus of typographical variations of popular domain names, which were ultimately verified by domain registration look-ups or automated web crawlers. Recent studies such as Szurdi *et al.* [110] examined a wider scope of popular domain names while Agten *et al.* [26] examined the nature of typosquatting over time. Rather than validating potentially typosquatted domain names through registration records, the most recent study by Khan *et al.* [64] introduced a novel approach called the *conditional probability model*. This model essentially identifies typosquatted domains by investigating which domains have high proportions of visitors leaving for more popular domains with lexically-similar names soon after landing.

The prior work described above has proven to be valuable in understanding the landscape of typosquatting. First, it highlights the prevalence of the problem with direct empirical evidence on how domain names are being typosquatted. Second, it quantifies the various techniques utilized by these adversaries for generating typographical variations of the domains they target. Additionally, several of these studies have paved the way for introducing countermeasures to combat the

problem. Defensive registration, for example, is perhaps the best countermeasure which involves domain name owners to register lexically-similar domain names to eliminate the opportunity for typosquatting altogether.

**Limitations.** While these previous studies have showcased the prevalence of various typosquatting techniques, they fail to account for certain realities. For example, examining DNS queries can produce very useful data through DNS tracing—but it does have limitations. First, domain name queries are not necessarily the result of actual queries by users, but rather from browser pre-fetching or automated web crawling. In addition, the typical analysis of DNS traffic was based on a short period of collection time, from which the identification of typosquatted domain names is as only good as the users who explicitly query for them.

Since DNS is inherently complex and diverse, it is difficult to exactly quantify the relevance of typosquatting techniques at any point in time due to acquiring DNS zone files for several domains. For example, obtaining the top-level domain (TLD) zone file for `.com` requires approval from Verisign [21] which updates them daily (at the time of this writing, 120,517 new domain names were added in the `.com` TLD alone [18]). This problem is further exacerbated with the adoption of the new gTLD program, which provides even more opportunities for registering domain names and thus the potential for typosquatting [10]. Even worse, the registration of a domain name is generally a low-cost activity—to the point where typosquatted domain names are disposable resources: once the adversary is done with the malicious activity, the domain names are most likely blocked or de-registered (within the grace period of registration; often up to 7 days allowed by registries). In conclusion, while there is fair amount of work on the problem at hand, none of such work measured how users behave when exposed to a typosquatted domain name.

**Contributions.** To ultimately address the issues described above, we chose to design and implement a user study which would provide several benefits: 1) it can be made scalable, 2) it can

provide insight into how various (theoretical) typosquatting techniques compare to each other, 3) it can determine why certain people fall for typosquatting while others do not (*e.g.* demographics), and 4) it can help discover the effectiveness of various techniques to mitigate the prevalence of typosquatting (*e.g.* security education and its benefits)—which could hopefully be useful for developing strategic defenses.

In this work, we design and evaluate a user study to gauge the effectiveness of several typosquatting techniques that are used “in the wild”. More specifically, we make the following contributions:

- (i) We validate typosquatting techniques presented in prior studies by examining their prevalence using various carefully sampled domains from several data sources.
- (ii) We experimentally demonstrate how security education and awareness of cybercrimes, particularly typosquatting, will affect the behavior of Internet users.
- (iii) We highlight various correlations between attributes of participating subjects and their proneness to accepting typosquatted domains, and hint on leveraging cognitive traits of Internet users to strengthen the defense against typosquatted domains.
- (iv) We publicly release our data so others can verify and build upon our research findings and results.

## 2.2 Background & Related Work

While typical Internet users may not have come across the term *typosquatting*, they may have inadvertently stumbled upon a typosquatted domain when they either typed the wrong URL into their web browser’s address bar or clicked on an external hyperlink. The actual term *typosquatting* was coined in the late 1990’s [51] to describe a new trend appearing alongside *cybersquatting*, a noto-

rious practice where opportunistic individuals preemptively registered domain names in the hopes of selling them back to companies and trademark owners for a substantial profit. Over the years, several studies have been conducted to understand models of typosquatting, including various features of typosquatted domain names. In the following, we review the technical anatomy of these typosquatting models and provide an overview of the various features prevalent in typosquatted domain names.

### 2.2.1 Identifying Typosquatted Domains

Prior studies conducted on typosquatting typically began their data collection phase by first identifying a set of domain names and then generating a list of possible typo variations on those domain names. Often these studies used a subset of the top-ranking domain names according to some domain ranking websites, such as Alexa. The rationale of using such domains is that typosquatters will naturally target the most popular domain names to increase the chances of obtaining unsuspecting visitors. Table 2.1 summarizes these several approaches which includes the authoritative domains they studied, the number of possible typosquatted domains they generated, and what percentage of them resolved to an actual web server’s IP address. In the following, we describe the models that generated typo variations of an authoritative domain.

**Typo-generation models.** One of the first and widely cited approaches for typo domain name generation was introduced in 2006 by Wang *et al.* [117], where the following five typo-generation models were used in the wild:

1. **Missing-dot:** this typo happens when omitting the dot after the “www” label, *e.g.* `wwwSouthwest.com`.
2. **Character-omission:** this typo happens when a character in the original domain name is



removed, *e.g.* `Diney.com` (a typo of the Disney brand).

3. **Character-permutation:** this typo happens when two consecutive characters are swapped in the original domain name, *e.g.* `NYTiems.com`.
4. **Character-substitution:** this typo happens when characters are replaced in the original domain name by their adjacent ones on a specific keyboard layout, *e.g.* `DidneyWorld.com`, where “s” was replaced by the QWERTY-adjacent character “d”.
5. **Character-duplication:** this typo happens when characters are mistakenly typed twice (where they appear once in the original domain name), *e.g.* `Googlle.com`.

Table 2.1: Summary of Typo Domain Identification Approaches.

Approach	Authoritative Domains	Typo Models	Domains Generated	Active Domains
Wang 2006 [117]	Alexa Top 10,000	(1) Missing-Dot	10,000	51% (5,094)
	Alexa Top 30	(1-5)	3,136	71%(2,233)
	MillerSmiles <sup>1</sup> Top 30	(1-5)	3,780	42%(1,596)
	Top 50 Children’s Sites	(1-5)	7,094	38%(2,685)
Keats 2007 [63]	Top 2,771 ( <i>Various Sources</i> )	(1-5)	1,920,256	7% (127,381)
McAfee Labs 2008 [49]	Top 2,000 ( <i>Unknown Source</i> )	<i>Unknown</i>	<i>Unknown</i>	80,000
Banerjee 2008 [31]	Top 900 ( <i>Various Sources</i> )	(6-8)	~3 million	35%
Moore 2010 [87]	Alexa Top 3,264	(1-5)	1,910,738	~49%(938,000)
Szurdi 2014 [110]	Alexa Top 1 million	(1-5)	~4.7 million	~20%
Agten 2015 [26]	Alexa Top 500	(1-5)	28,179	61% (17,172)

While this previous study presented the first attempt to systematically understand the most prevalent typosquatting techniques based on certain usage aspects, later studies looked at exhaustively

<sup>1</sup>[www.MillerSmiles.co.uk](http://www.MillerSmiles.co.uk) is one of the web’s largest collection of phishing scams and scam emails.

generating typo domains using other methods. For example, a similar approach in 2008 by Banerjee *et al.* [31] suggested the following for generating typosquatted domains:

6. **1-mod-inplace:** the typosquatter substitutes a character in the original domain name with all possible alphabet letters.
7. **1-mod-inflate:** the typosquatter increases the length of a domain name (or URL) by one character. Unlike in [117] characters are added based on distance (*e.g.* using a keyboard layout), this work considers all characters as potential candidates.
8. **1-mod-deflate:** similar to the approach in [117], this typo happens when a typosquatter removes one character from the original domain name (or URL).

Certain aspects of the techniques proposed in [31] can be viewed as generalization of the techniques proposed in [117]. For example, rather than substituting adjacent characters on a keyboard as shown by Wang *et al.*'s fourth model, Banerjee *et al.* substituted all possible alphabet characters when generating typo domains. In addition, they also experimented with two and three character modifications for their **inplace**, **inflate** and **deflate** schemes thereby generating roughly three million possible typo domain names starting with a corpus of 900 original domain names.

After probing for the existence of a possible typo domain, Banerjee *et al.* observed that approximately 99% of the “phony” typosquatted sites they identified utilized a one-character modification of the popular domain names they targeted. Essentially, these are domain names that have a Damerau-Levenshtein distance [44, 76] of one from the domains they target. The Damerau-Levenshtein distance is “the minimum number of operations needed to transform one string into another, where an operation is defined as an insertion, deletion, or substitution of a single character, or a transposition of two adjacent characters” (a generalization of the Hamming distance).

## 2.2.2 Features of Typosquatted Domains

In the following, we review features of typosquatted domain names as confirmed by measurements and their evolution over time, including length of domain names (2.2.2.1), popularity of domain names (2.2.2.2), popularity of top-level domain (TLD) (2.2.2.3), and domain landing behavior (2.2.2.4).

### 2.2.2.1 Domain Name Length

While investigating if domain name length affects the chances of being typosquatted, Banerjee *et al.* [31] observed that more than 10% of all possible “phony” typosquatted sites registered on the Internet have URLs with less than 10 characters. This fulfills their expectation that typosquatters target domains with shorter names, since popular sites often have short names. However, in a contradictory study by Moore and Edelman [87], the authors show that “no matter the length of the popular domain, typo domains within the Levenshtein or fat-finger distance 1 of popular domains were overwhelmingly confirmed as typos.” Naturally, we can expect that as the length of domain names increases the probability of it being typosquatted increases, since the number of possible typo variations increases. This concept is solidified in the results of the 2015 study by Agten *et al.* [26], which concluded that “typosquatters have started targeting longer authoritative domains in the past six years (from 2009)”, due to the fact that typosquatted domain names with short lengths were already registered. Our study, on the other hand, confirms that users are equally likely to fall for typosquatted domains regardless of their length.

### 2.2.2.2 Domain Name Popularity

Another feature of domain names that has been investigated for its correlation with typosquatting is their popularity. It is naturally expected that typosquatters will target the most popular domain names to maximize the return on their investment (*i.e.* the number of visits by unsuspecting users). The results of Banerjee *et al.* [31] initially suggest that “the percentage of active typosquatting domains for a given authoritative domain reduces significantly with the decreasing popularity.” This is in contrast to the results presented by Szurdi *et al.* [110], who performed a comprehensive study of typosquatting domain registrations within the `.com` TLD—the largest TLD in the domain name ecosystem. They concluded that 95% of typo domains target the “long tail” of the popularity distribution. The longitudinal study by Agten *et al.* [26] also confirms this trend, suggesting a shift in trends and behaviors of typosquatters.

### 2.2.2.3 Effect of the Top-Level Domain

The popularity of a TLD has been also investigated as a feature for its correlation with typosquatted domain names. For example, since the `.com` TLD was introduced as one of the first TLDs when the Domain Name System (DNS) was first implemented in January 1985 [12], it makes up a large portion of the total number of registered domain names—As of June 30, 2015, the total number of registered domain names was 294 million, out of which 117.9 million domain names were registered under `.com`, making up roughly 40% of the total domain names (<http://bit.ly/1VKiMr3>). As such, a majority of the existing studies conducted on typosquatting have only considered domain names in the `.com` TLD. In their results, Banerjee *et al.* [31] observed that for nearly a quarter of all initial `.com` URLs, at least 50% of all possible phony sites exist; confirming that a domain name ending with `.com` has a high chance of being typosquatted. Interestingly, the results of Agten *et al.* [26] finds that certain country-code TLDs (`.uk`, `.jp`, etc.) affect

the number of typosquatted domains they contain due to either an unconventional domain dispute policy or domain cost (*i.e.* cheaper domain names are more likely to be typosquatted).

Additionally, the TLD portion of a domain name may also be a target for exploitation. As mentioned previously, typosquatters have targeted popular `.com` domain names by registering a similar domain name in the country code TLD (ccTLD) for Oman (*e.g.* `Netflix.om`). Furthermore, `.com` domains may have a malicious `.org` counterpart unbeknownst to the original registrant of the `.com` domain. A noteworthy example of this was mentioned in [41], where unsuspecting viewers inadvertently typed `www.whitehouse.com` instead of `www.whitehouse.gov` and were exposed to questionable content in lieu of the official United States White House website. Banerjee *et al.* [31] further studied this effect and observed that: “1) Original URLs with a `.com` extension are impersonated primarily in `.biz`, `.net` and `.org` domains. 2) Original URLs without a `.com` extension are impersonated primarily in `.com`, `.net` and `.org` domains”.

#### 2.2.2.4 Probability Models for Domain Landing

The 2015 study by Khan *et al.* [64] introduced a novel approach for detecting typosquatting domains called the *conditional probability model*, which requires “a vantage point at the network level to examine DNS and HTTP traffic records”. This model identifies domains that have a high proportion of visitors that navigate to similar-looking popular domains after having landed on a particular site (domain name). Specifically, they generated “pairs of domains ( $d_1, d_2$ ) such that each load was performed within 33 seconds of each other, and the Damerau-Levenshtein edit distance between the two domains is one”. When dealing with lexically-similar domain pairs, where one of the two domains is unlikely a typo of another, *e.g.* `nhl.com` and `nfl.com`, the advantage of applying the conditional probability model is that it does not correlate such domain pairs. In the results reported by Khan *et al.*, “a request for `nhl.com` is only followed by a load of `nfl.com`

.08% of the time where the reverse rate is even lower at  $< 0.01\%$ ". However, they reported that users attempting to visit `eba.com` will immediately navigate to `ebay.com` 90% of the time, which indicates that `eba.com` might be a typoquatted domain.

## 2.3 User Study: Identifying Typo Domains

To gauge the effectiveness of typosquatting techniques discussed in Section 2.2.1 that are prevalent in the wild, we conducted a user study. Subjects were presented with a list of actual domain name URLs where a subset of them were modified using all of the techniques shown in Section 2.2.1 to represent possible typosquatted domain names. The subjects were asked to indicate that for each given domain name URL, select "Yes" if it appears to be a typosquatted domain name or "No" if it is an authoritative domain name.

### 2.3.1 User Study Objectives

The primary objectives of the user study are to:

- (i) Gauge the effectiveness of various techniques of typosquatting on users.
- (ii) Study the benefits on how security education can improve users' awareness of typosquatted domain names.

Secondary objectives include:

- (iii) Understanding correlations between user demographics and the outcomes of typosquatting (whether they fall for it or not).
- (iv) Determining features of successful typosquatted domains.

In particular, we hope to answer the following questions:

- Q1. Are users more susceptible to typosquatted domain names containing certain kinds of typos (*e.g.* missing characters) than others (*e.g.* substituted characters)?
- Q2. Does security education play a role in helping users correctly identify typosquatted domain names?
- Q3. Does a user's demographic (*e.g.* age, education) affect how they perceive typosquatted domain names?
- Q4. Do users *more easily* identify typosquatted domain names that target popular domains?
- Q5. Are certain types of typosquatted domain names (*e.g.* alphanumeric) more susceptible than others?
- Q6. Does the TLD (*e.g.* .com, .uk) affect a user's identification of a typosquatted domain name?

To answer these questions and achieve our objectives, we rely on a systematic method for the selection of domains and subjects, as well as experimental design and evaluation criteria. In the following, we elaborate on each of those aspects.

### 2.3.2 *Selection of Domain Names*

The list that was presented to each subject comprised of 200 domain names that were chosen from the Alexa top 1 million websites (globally). Rather than sample domain names randomly, the entire collection of 1 million domain names were split into four unequal partitions with the first partition representing the top 1,000 domain names. Subsequent partitions were increased in size by a factor of 10 and then 50 domain names were randomly sampled from each partition to bring us a total

of 200 candidate domain names. This method allowed us to favor domain names that appeared towards the popular end of the spectrum.

Next, we simply iterated through the candidate domains and randomly decided if it should be typosquatted or not. If chosen to be typosquatted, a candidate domain is then modified using one of the techniques from Section 2.2.1, which was also randomly chosen. Ultimately, 93 (46.5%) candidate domains were randomly chosen to be typosquatted, with Table 2.2 showing a breakdown of how many candidate domains were modified using a particular typosquatting model. Since *Model 8* (1-mod-deflate) is similar to *Model 2* (Character-omission), we only considered *Models 1-7* in our study.

Table 2.2: Number of candidate domains per model

<b>Typosquatting Model</b>	<b># of Domains</b>
1 (missing-dot)	15
2 (character-omission)	11
3 (character-permutation)	12
4 (keyboard-substitution)	11
5 (character-duplication)	19
6 (1-mod-inplace)	12
7 (1-mod-inflate)	13
<i>Total Domains Modified:</i>	93

### 2.3.3 Selection of Subjects

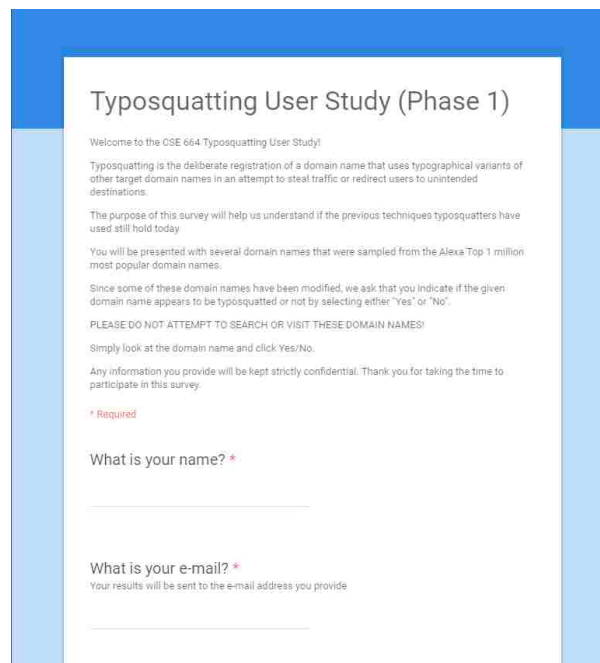
The participants of the study primarily consisted of University students, followed by University staff and researchers. Despite the lack of choice in participants, we strove to include a good representation of demographics that would address the questions raised in our study’s objectives (*i.e.*



Does a user's demographic affect how they perceive typosquatted domain names?). Additionally, we attempted to include diverse sample characteristics (with respect to subjects) so that we can understand other objectives of the study (e.g. whether security education, familiarity, or educational background, help identify typosquatted domain names).

## 2.4 Design of the User Study

To assess how prior knowledge and awareness of security concepts affect a user's behavior, the user study encompassed three phases which incrementally introduced subjects to all of the typosquatting techniques discussed in Section 2.2.1. To deploy the user study, we created an online survey form so each participant could complete the survey anytime they wish using their own computers and devices.



The image shows a screenshot of a web-based survey form titled "Typosquatting User Study (Phase 1)". The form is enclosed in a blue border. The text on the page reads:

**Typosquatting User Study (Phase 1)**

Welcome to the CSE 664 Typosquatting User Study!

Typosquatting is the deliberate registration of a domain name that uses typographical variants of other target domain names in an attempt to steal traffic or redirect users to unintended destinations.

The purpose of this survey will help us understand if the previous techniques typosquatters have used still hold today.

You will be presented with several domain names that were sampled from the Alexa Top 1 million most popular domain names.

Since some of these domain names have been modified, we ask that you indicate if the given domain name appears to be typosquatted or not by selecting either "Yes" or "No".

PLEASE DO NOT ATTEMPT TO SEARCH OR VISIT THESE DOMAIN NAMES!

Simply look at the domain name and click Yes/No.

Any information you provide will be kept strictly confidential. Thank you for taking the time to participate in this survey.

\* Required

What is your name? \*

\_\_\_\_\_

What is your e-mail? \*

Your results will be sent to the e-mail address you provide

\_\_\_\_\_

Figure 2.1: Phase 1 of the malicious domain name user study.

For Phase One (Fig. 2.1), participants were given a URL link to the online survey website and were presented with a brief introduction to typosquatting (omitting any of the previously-discussed techniques from Section 2.2.1). Following the introduction section, a series of questions were included to gather the following demographical data: *Name, E-mail, Gender, Age, Education, and Familiarity of Security Concepts* (on a scale 1-5). Before participants could proceed to the next page of the online survey, they must enter the current time (this was necessary in order to calculate the amount of time each subject completed the survey). Each subsequent page of the survey presented 10 candidate domain URLs with a “Yes” or “No” choice to indicate a typosquatted domain.

Phases Two and Three (Fig. 2.2) followed a similar template as Phase One, except they only asked for the e-mail provided in Phase One (to uniquely identify subjects) and the current time. Additionally, the same corpus of candidate domain name URLs used in Phase One are shown *except the order in which they appear are randomly shuffled*. To provide subjects with more knowledge about typosquatting techniques, the introduction sections of Phases Two and Three include typo-generation models mentioned in Section 2.2.1.

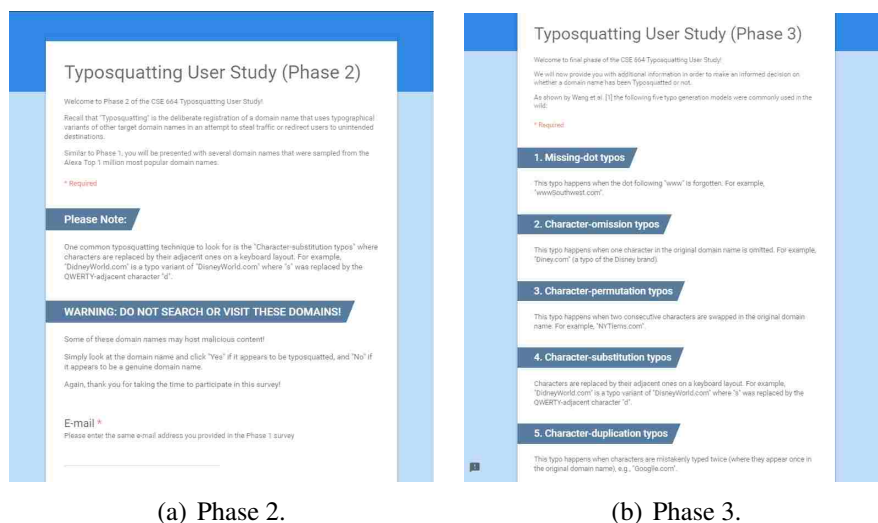


Figure 2.2: Screenshots of phases 2 & 3 of the malicious domain name user study. Notice that each phase introduces the participants to increasing levels of knowledge about typosquatting.

## 2.5 Results & Discussion

In our user study, we recruited 34 participants who completed all three phases of the survey over a one week period. After completing each phase of the survey, each participant received an automated e-mail containing the total number of correct responses (out of 200) which forms their score.

### 2.5.1 Evaluation Criteria

For our evaluation, we primarily observed two performance metrics among users: 1) correct responses and 2) the amount of time to complete each phase of the study. For a given domain name, a correct response is defined as to whether the user answered “No” if the given domain name was an authoritative domain name (*unaltered*) or “Yes” if the given domain name was indeed typosquatted (altered according to Section 2.2.1). We examined the total completion time for each user and calculated the average amount of time spent (in seconds) to answer each question of the 200-question survey.

As will be shown in the next section, users generally performed better (*i.e. correctly identified typosquatted domain names*) with each subsequent phase of the survey. However, if we drill down and examine the users’ demographical data, we can see that variables such as their *Age* and *Education* affect not only how they perceive potentially typosquatted domain names—but also how long they spend analyzing them. These interesting findings are discussed further in our demographical results outlined in Section 2.5.3.

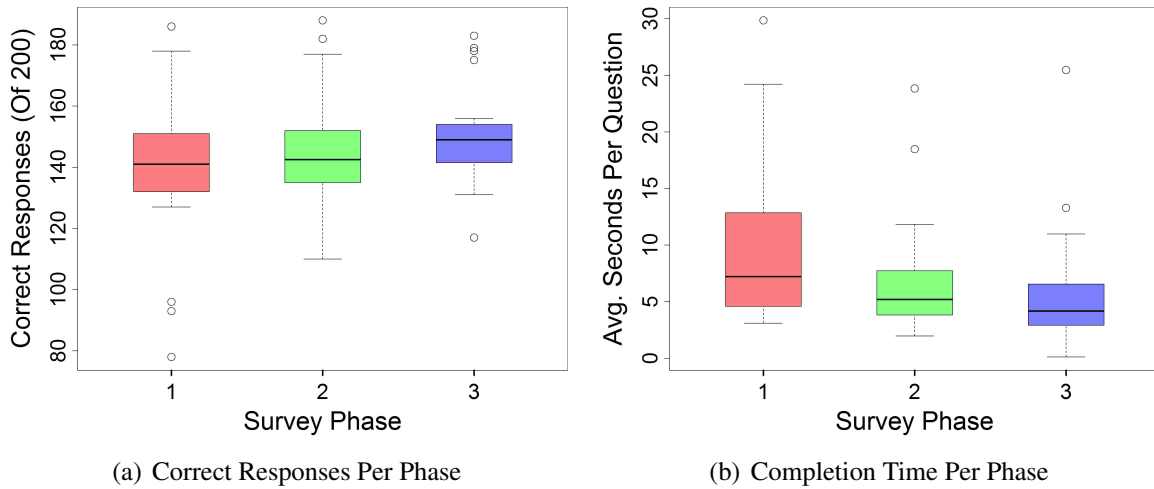


Figure 2.3: Plots of the study results depicting correct responses and completion time per phase.

### 2.5.2 Participant Scores and Completion Time

As illustrated in Figure 2.3(a), the average number of correct responses improved with each subsequent phase. For Phase One, the scores ranged from 78 to 186 correct responses with a mean, standard deviation and variance of 142.2, 23.6 and 557.1, respectively. In Phase Two, the minimum score increased to give us a range from 110 to 188 (Mean=147.1, s.d.=18.6, variance=345.7). Phase Three's minimum score increased slightly to range of 117 to 183 (Mean=149.9, s.d.=15, variance=225).

As depicted in Figure 2.3(b), the average number of seconds per response decreases slightly with each phase. For Phase One, the average elapsed time per response ranged from 3.1 to 29.8 seconds (Mean=9.6, s.d.=6.7, variance=44.5). In Phase Two, the minimum average time per response decreased to give us a range from 1.9 to 23.8 seconds (Mean=6.7, s.d.=4.7, variance=21.8). Phase Three's minimum score also decreased slightly to range of 0.1 to 25.5 (Mean=5.5, s.d.=4.6, variance=20.8).

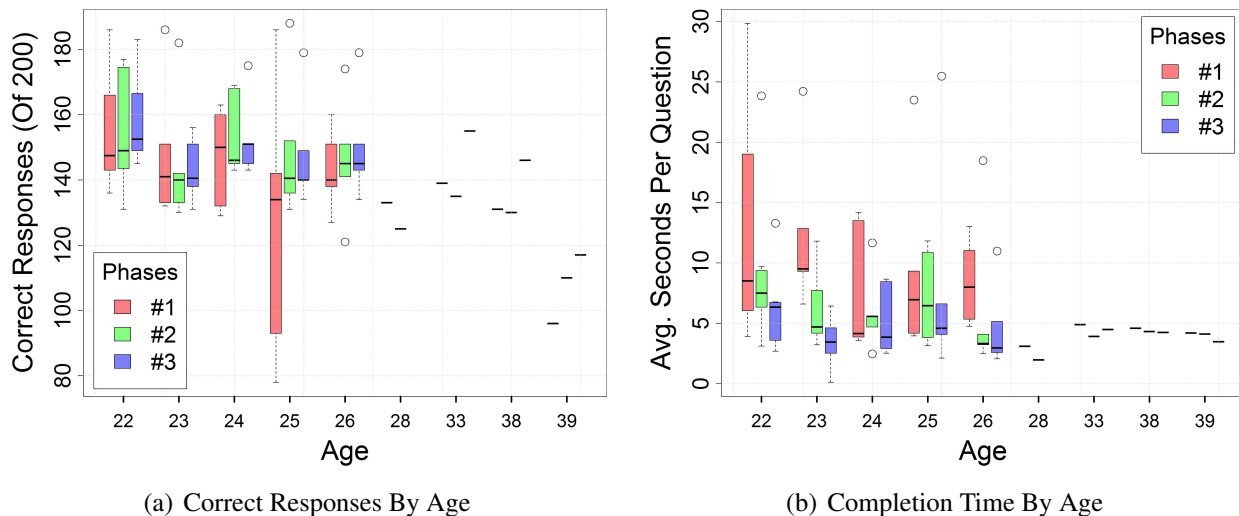


Figure 2.4: Plots of the user study results according to age.

Those results are interesting in several ways. First, the more subjects were educated about the security problem at hand, the faster they became at identifying typosquatted domains. Second, subjects also became better at identifying those domains (*i.e.* better identification rate).

### 2.5.3 Demographics

**Age.** The ages of the participants ranged from 22 to 39 (Mean=25, s.d.=4.1, variance=16.5). As we can see in Figure 2.4(a), younger participants generally scored higher than older participants across all phases of the study. Interestingly, Figure 2.4(b) shows that even though the younger participants scored higher, they also spent more time per question on average compared to their older counterparts.

In the fields of Psychology and Optometry, it is generally understood that older adults often take longer to read than young adults. As Paterson *et al.* [91] points out, this age-related difference is “due to optical changes and changes in neural transmission that occur with increasing age”.

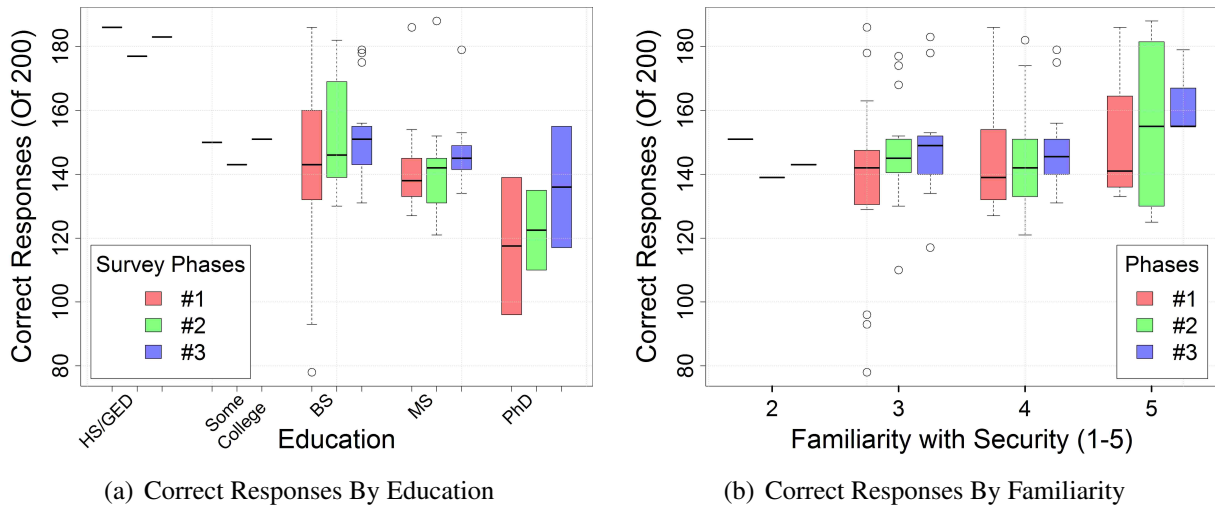


Figure 2.5: Plots of the user study results according to education and familiarity

Conversely, the results of our study show a trend where older participants spent less time per survey question on average than their younger counterparts. This trend of spending more time on each question in the survey could explain why the younger participants scored better, as the older participants appeared less patient and tended to perform worse at identifying typosquatted domain names.

**Education level.** The majority of participants were University students with University staff making up the rest of the test subjects. Of the participants, there was only 1 High School Graduate and 1 participant who reported some College Education. For the rest, 17 participants (50%) had a Bachelors degree, 13 participants (38.2%) had a Masters degree, and 2 held Doctorate degrees (5.88%). As shown in Figure 2.5(a), participants holding higher degrees of education actually scored worse than participants with less education.

**Familiarity of security concepts.** On a scale of 1 to 5 in the familiarity of security concepts, only 1 participant chose a value of “2”. 15 participants (44.1%) chose the middle value of “3”,

14 participants (41.1%) chose the higher value of “4”, and the remaining 4 participants (11.8%) stated they were very familiar with security concepts by choosing “5”. As we can see in Figure 2.5(b), the self-identification of one’s familiarity with security concepts coincides with how well they performed as scores generally increased.

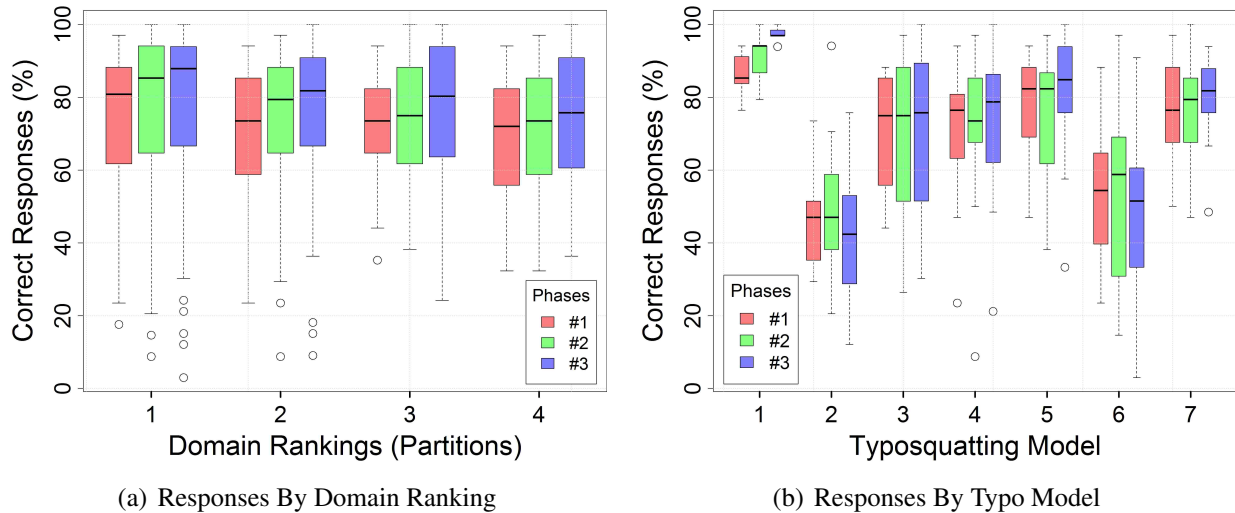


Figure 2.6: Plots of the user study results according to domain ranking and typo model.

#### 2.5.4 Domain Name Features

**Domain ranking.** As mentioned previously, the entire collection of 1 million domain names were split into four unequal partitions with *Partition 1* representing the top 1,000 domain names. Subsequent partitions were increased in size by a factor of 10, so *Partition 2*, *Partition 3*, and *Partition 4* represented the ranges: [1,001-10,000], [10,001-100,000], and [100,001-1,000,000], respectively. As expected, Figure 2.6(a) demonstrates that participants were more successful in correctly identifying typosquatted domain names that targeted popular domains.

**Typosquatting model.** Since *Model 8* (1-mod-deflate) is similar to *Model 2* (Character-omission), we only considered *Models 1* through 7 in our study. As shown in Fig. 2.6(b), participants were very likely to distinguish a typosquatted domain name that used *Model 1* (Missing-dot), *Model 5*

(Character-duplication) and *Model 7* (1-mod-inflate). The models that caused most participants to incorrectly identify typosquatted domain names were *Model 2* (Character-omission) and *Model 6* (1-mod-inplace). Essentially, users tend to correctly identify typosquatting which adds characters (e.g. duplicate or random) while *the most effective typosquatting involves permutations and substitutions*.

To ascertain why certain typosquatting techniques are more effective than others, we can look at studies in Cognitive Science. For example, Grainger and Whitney [53] highlight the fact “that a text composed of words whose inner letters have been re-arranged can be can be raed wthi quatie anazing esae!” This so-called “jumbled word effect” has to do with how our brains perceive the placement of characters in words.

**Domain Name type.** Given our sample size of 200 domain names, 167 (83.5%) were made up of all alphabetic characters with the remaining 33 (16.5%) containing alphanumeric characters. Fig. 2.7(a) illustrates that participants were more likely to identify a domain name that contained all alphabetic characters as opposed to alphanumeric characters.

**TLDs.** According to the Internet Assigned Numbers Authority (IANA), the organization who delegates administrative responsibility of TLDs (Top-Level Domains), there are different “groups” of TLDs which include [14]:

- Infrastructure top-level domain (ARPA)
- Generic top-level domains (gTLD)
- Restricted generic top-level domains (grTLD)
- Sponsored top-level domains (sTLD)
- Country code top-level domains (ccTLD)
- Test top-level domains (tTLD)



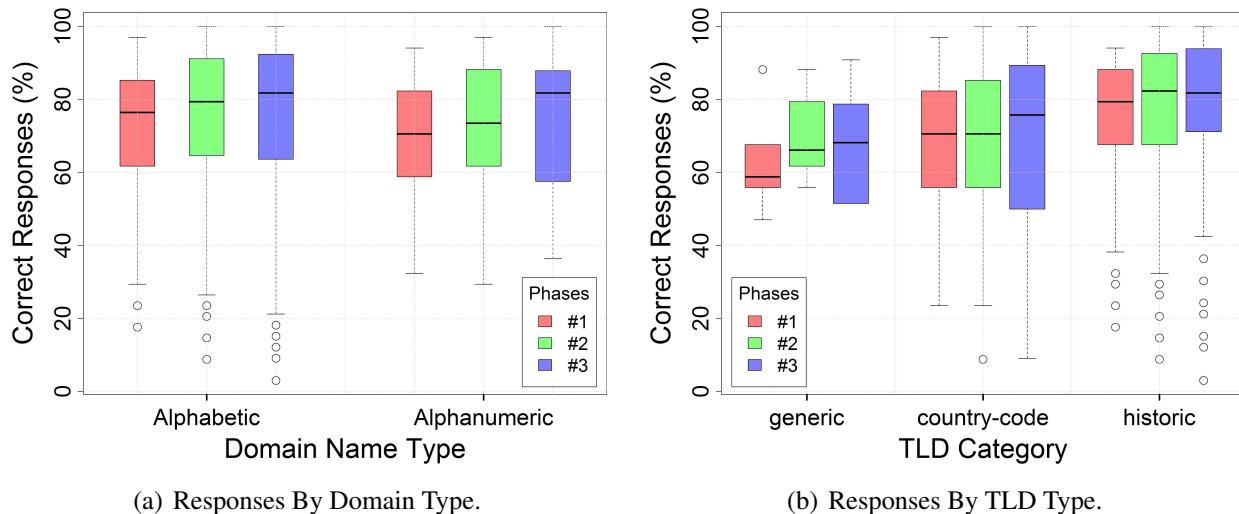


Figure 2.7: Plots of the user study results according to domain type and TLD.

The initial set of TLDs that were created in the early development of the Internet (*e.g.* `.com` and `.net`) are now grouped under the “generic” category mentioned above. For our purposes; however, we labeled the following TLDs under the “historic” group, since they are widely-known to the average Internet user: `.com`, `.org`, `.net`, `.int`, `.edu`, `.gov`, and `.mil`. As a result, our sample size of 200 domain names were only categorized into three groups where 119 (59.5%) fell into the “historic” TLD group, 75 (37.5%) fell into the “country-code” TLD group, and 6 (3%) were in the “generic” TLD group. As illustrated by Fig. 2.7(b), participants performed better when presented with a domain name with a “historic” TLD than domain names from either the “generic” or “country-code” groups.

**Typosquatting difficulty.** Table 2.3 lists the top 10 domains which caused the most incorrect responses (averaged over all phases) for the participants, which coincides with our earlier statement that the most effective techniques involve permutations and substitutions.

The first-most incorrectly identified site could be attributed to the fact that most participants were

based in the United States and therefore unfamiliar with the .ua TLD, which is the ccTLD for Ukraine. However, it should be pointed out that the fourth-most incorrectly identified domain name, `umblr.com`, turned out to be an edge case that is in fact an actual typosquatted domain. Most participants most likely thought it was a typosquatted variant of `tumblr.com`, a popular microblogging and social networking website. During the study design phase, our algorithm selected the domain `umblr.com` but did not actually modify it with a typosquatting model and subsequently marked it as *not* typosquatted. According to WHOIS records, the domain name `umblr.com` is actually owned by “Tumblr, Inc.” which makes this a perfect example of a defensive registration against potential typosquatters.

Table 2.4 lists the top 10 domain names that were correctly identified (averaged over all phases) by the participants of the study. Again, this coincides with that fact that users easily spot typosquatted domain names which adds characters—especially if they target popular domain names such as `google.*` or `blogger.com`.

Table 2.3: Top 10 incorrectly identified domains (unmodified domains shown in gray).

<b>Typo Domain</b>	<b>Authoritative Domain</b>	<b>Typosquatting Model</b>	<b>Rank</b>	<b>Phase 1 Correct</b>	<b>Phase 2 Correct</b>	<b>Phase 3 Correct</b>	<b>Average Correct</b>
---	onlainfilm.ucoz.ua	---	6,345	24%	9%	9%	14%
ngbus.com	tgbus.com	6 (1-mod-inplace)	998	24%	15%	3%	14%
afg.com	avg.com	4 (keyboard-sub)	366	24%	9%	21%	18%
---	umblr.com	---	506	18%	26%	15%	20%
egadget.com	engadget.com	2 (char-omission)	403	29%	21%	12%	21%
vc.cn	ivc.cn	2 (char-omission)	1,778	29%	24%	18%	24%
zasgames.com	oasgames.com	6 (1-mod-inplace)	7,942	32%	29%	15%	26%
hispress.com	hespress.com	6 (1-mod-inplace)	536	41%	26%	24%	31%
---	05tz2e9.com	---	5,988	32%	29%	36%	33%
rudupoint.com	urdupoint.com	3 (char-permutation)	443	44%	26%	30%	34%

Table 2.4: Top 10 correctly identified domains (unmodified domains shown in gray).

<b>Typo Domain</b>	<b>Authoritative Domain</b>	<b>Typosquatting Model</b>	<b>Rank</b>	<b>Phase 1 Correct</b>	<b>Phase 2 Correct</b>	<b>Phase 3 Correct</b>	<b>Avg. Correct</b>
googlje.dz	google.dz	7 (1-mod-inflate)	370	97%	100%	94%	97%
---	wayfair.com	---	568	94%	97%	100%	97%
blogger.comm	blogger.com	5 (char-dup)	72	91%	97%	100%	96%
---	office.com	---	63	94%	94%	100%	96%
---	audible.com	---	840	94%	100%	94%	96%
wwweromode.net	eromode.net	1 (missing-dot)	697,652	94%	94%	97%	95%
---	popsugar.com	---	837	91%	94%	100%	95%
syosetu.comm	syosetu.com	5 (char-dup)	930	94%	97%	94%	95%
wwiklan-oke.com	iklan-oke.com	1 (missing-dot)	688,829	91%	97%	94%	94%
financial-spread-bettin.gcom	financial-spread-betting.com	3 (char-perm)	729,388	85%	97%	100%	94%

## 2.6 Conclusions & Recommendations

This study has allowed us to gain valuable insight into the effectiveness of various typosquatting techniques and how security education affects the behavior of users. Our results confirm that participants generally performed better and faster at identifying typosquatted domain names *after* being educated about typosquatting models between each phase of the study.

**Recommendations.** Based on our results, we offer some recommendations for strengthening the defenses against typosquatting.

As demonstrated by the improved scores with each subsequent phase of the study, we can confidently say that thoroughly educating users about all known typosquatting techniques will surely help them fend off against malicious domain names. As more organizations and businesses adopt security training programs in this day and age, it would be most beneficial to incorporate a training module that specifically explores typosquatting in more detail (perhaps alongside the commonly-

taught *Phishing* attacks).

The results of the study, pertaining to the particular features of the domain names that were used, can most certainly aid in the design of a defense system that uses heuristic analysis. While typical defense systems use blacklists (*e.g. Google Safe Browsing*), a heuristic-based defense system that dynamically analyzes URLs can incorporate our findings to help “rank” potentially malicious domains. For example, domain names from a gTLD or ones with alphanumeric characters can be “flagged” for closer inspection since users are more likely to fall victim to their typosquatted variants.

## CHAPTER 3: MALICIOUS & TYPOSQUATTED DOMAIN NAME ATTACKS WITH CONTEXT

In Chapter 2, we presented an overview of current state-of-the-art typosquatting techniques and how effective they were in deceiving users. While some techniques fared better than others (*e.g.* swapping characters *vs.* adding characters), such typo generation techniques fail to take the context of the user into account. As evidenced by the results of the user study presented in Section 2.5.4, the most incorrectly identified domain name in the study was likely attributed to a user’s geographical setting as most were unfamiliar with domains ending with a ccTLD for Ukraine (.ua).

In this chapter, we explore the possibility of taking context into account when devising an attack strategy involving malicious & typosquatted domain names. Some of these approaches may include: 1) profiling a user by eavesdropping on their network activity (*e.g.* HTTP or DNS traffic) and 2) profiling a user by co-locating on the same host (*e.g.* extracting browsing history, etc). The motivation behind this approach is to determine if an attacker can improve their “success” rate of utilizing such malicious domain names. In turn, this will lead into an initial study of the effectiveness of such advanced attacks which will help aid in the development of possible countermeasures.

### 3.1 Background & Related Work

As mentioned previously, we can profile a user by eavesdropping (*i.e.* network snooping or packet sniffing) on their network activity. This is typically accomplished by capturing TCP/IP or other protocol packets and decoding the contents using a protocol analyzer or similar tool such as *Wireshark*. However, as Laboshin *et al.* [73] state: “most of these tools are designed to run on single high-performance servers which are not capable of handling huge amounts of traffic data

captured at very high-speed”. Even the official *Wireshark Guide* [15] warns that using the tool with a high-traffic network will generate large capture files: “Capturing on a gigabit or even 100 megabit network can produce hundreds of megabytes of capture data in a short time”. Given the potentially large volumes of network traffic, if one was to do HTTP eavesdropping then the main challenge would be to know what is relevant and what is not. As Campbell points out [92], traditional packet capture tools cannot keep up unless you know exactly where to look: “*Relying only on traditional packet capture would be like using a scalpel to cut down a tree*”.

### 3.1.1 The DNS Protocol

Alternatively, we can focus on the DNS protocol to conduct user profiling since DNS requests and responses are typically small (originally capped at 512 bytes as a requirement, today under 1500 bytes as a practical matter) [127] and usually sent “in the clear” (*i.e.* unencrypted)[99]. In choosing the DNS protocol for eavesdropping, there are yet additional challenges to consider such as: 1) DNS resolution on shared hosting entities (which are not of particular interest to the users) and 2) DNS prefetching, which could affect the accuracy of profiling users since certain browsers attempt to pre-resolve the domains found in hyperlinks when loading webpages. Despite these challenges, an adversary who is determined can still profile users to a high degree by just using standard eavesdropping tactics out-of-the-box.

Domain names essentially allow us to map human-readable strings to machine-readable IP addresses. In the case of IPv4, the address on the network is composed of 4 bytes (32 bits) in total, and can be represented by four number segments separated by dots as in 1 . 2 . 3 . 4. IPv6, which has an address space that is four times larger than IPv4, has a total of 8 segments in its address structure, with each segment being 2 bytes that are represented by hexadecimal numbers (*e.g.* 2a03:2880:f10c:83:face:b00c:0:25de). It is a herculean task for users to remember all the

numerical addresses of the Internet services they want to access. DNS, which is designed to improve the usability, allows users to access them through the familiar natural language, such as `www.example.com`.

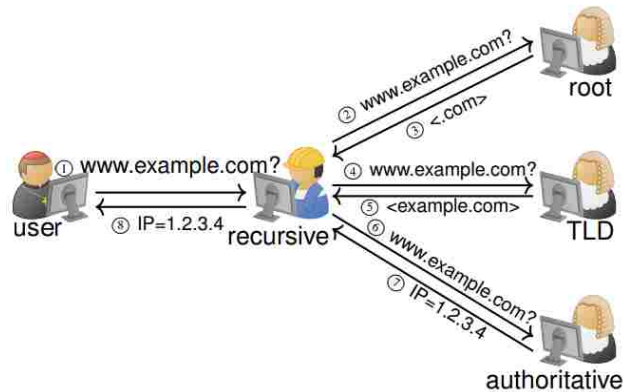


Figure 3.1: An illustration of the steps of DNS resolution through the recursive, root, TLD, and authoritative name server.

### 3.1.2 DNS Resolution

DNS is a hierarchical and distributed database structure for resolving domain names. The string address entered by users is converted to an IP address through root, Top-level domain (TLD), and authoritative name servers. Figure 3.1 shows the translation of the domain name. When users attempt to access a particular service through an Internet web browser such as Chrome or Firefox, the DNS resolution process begins.

**Local cache and host table.** As the first step of DNS resolution, the local resolver initially finds its local cache and host table. If there is no corresponding entry for the given domain, the local resolver generates and sends the DNS query to the recursive server for the DNS resolution. Step ① in Figure 3.1 shows the initial DNS query from the client to the recursive server. The query

includes a special flag to indicate that is a recursive query. Once the recursive server receives the query from the client, it begins the recursive steps ② through ⑦ with the root, TLD, and authoritative name servers.

**Root name server.** In step ②, the recursive name server (which does not have a corresponding entry for the received query from the client) sends a iterative (*i.e.* non-recursive) query to the root name server. This server is responsible for the translation of the root zone in the DNS system using stored data, such as the IP address and location of an authoritative TLD name server. Upon receiving the query of `www.example.com` in step ③, the root name server returns the appropriate list of authoritative TLD servers for the `.com` TLD.

**TLD name server.** The next phase, step ④, is querying for the translation of the TLD in given domain. The resolver that found the TLD name server list through the previous step sends the domain name query to the `.com` TLD name server. The `.com` TLD name server searches for the record of the authoritative name server corresponding to the queried domain and responds to the recursive server with a list as shown in step ⑤. For example, if `example.com` has two authoritative name servers, namely `ns1` and `ns2`, the information about `ns1.example.com` and `ns2.example.com` would be returned as well.

**Authoritative name server.** The final step of recursive domain name resolution process goes through the authoritative name server. In step ⑥, the recursive name server contacts the name server from the records received from the `.com` TLD name server. The authoritative name server, which knows the A address record for `www.example.com` (or AAAA address record in the case of IPv6), sends the result to the recursive server in step ⑦. At this point, the recursive name server determines the IP address of the requested domain name and forwards it to the local machine shown in ⑧. The local resolver, which finally knows the IP address of `www.example.com`, delegates it to a web browser. As a result, the web browser will be able to initiate the loading of a web page



by sending a Hypertext Transfer Protocol (HTTP) request to the resolved IP address.

### 3.1.3 DNS Prefetching

As described in the previous section, there are several steps in the process of resolving a domain name which can cause a considerable amount of latency. As the *Chromium* documentation [2] explains: “The time that DNS resolution takes is highly variable, where latency delays can range from around 1ms (locally cached results) to commonly reported times of several seconds.” This prefetching attempts to alleviate this latency by resolving domain names automatically to help speed up Internet browsing.

For example, when the open-source Chromium browser encounters hyperlinks in web pages, it parses the domain name from each link and resolves them to an IP address [2]. When a user clicks on any of these pre-resolved domains, the Chromium team claims that this saves on average about 200 milliseconds (assuming they have not already visited the domain recently) and avoids the “worst case” delay, which are regularly over 1 second. In addition to resolving domain names found in links of web pages, Krishnan and Monrose [71] point out that browsers such as Chrome “will attempt to guess the site a user might be attempting to visit as she types in the location bar, simultaneously performing pre-resolutions for the predicted destinations”.

### 3.1.4 Establishing Context

**Eavesdropping.** As mentioned earlier, DNS queries and responses are usually sent unencrypted and “in the clear” [99]. While the original DNS specification [90] supports both UDP and TCP, UDP is the recommended method for standard queries over the Internet while TCP is typically used for zone transfers. Because of the connection-less nature of UDP, Zhu *et al.* [128] argue

that the use of this protocol in DNS “is the root cause of a range of fundamental weaknesses in security and privacy that can be addressed by connection-oriented DNS”. To this end, an adversary wishing to eavesdrop on DNS traffic will glean much information if they intercept unencrypted UDP packets at certain vantage points in the DNS resolution process shown in §3.1.2. According to Bortzmeyer [99], “the best place to setup a network tap, from an eavesdroppers point of view, is clearly between the stub resolvers and the recursive resolvers” (see Figure 3.1), because traffic is not limited by DNS caching.

**Co-Locating on the Same Host.** Our threat model assumes that an adversary has access to the same host as the target user, such as a shared workstation or public computer at a library or school. These systems will most likely be locked down by an administrator with little to no privileges. If such shared computers were unrestricted, or if the adversary manages to circumvent these restrictions, then this scenario would fall outside the scope of our threat model since that opens up the possibility for a wider range of attacks (*e.g.* malware, etc.). To that end, we are most concerned with the aspect of privacy and whether it is possible to access information on a shared host about its users in order to accurately profile them to launch more sophisticated and successful attacks.

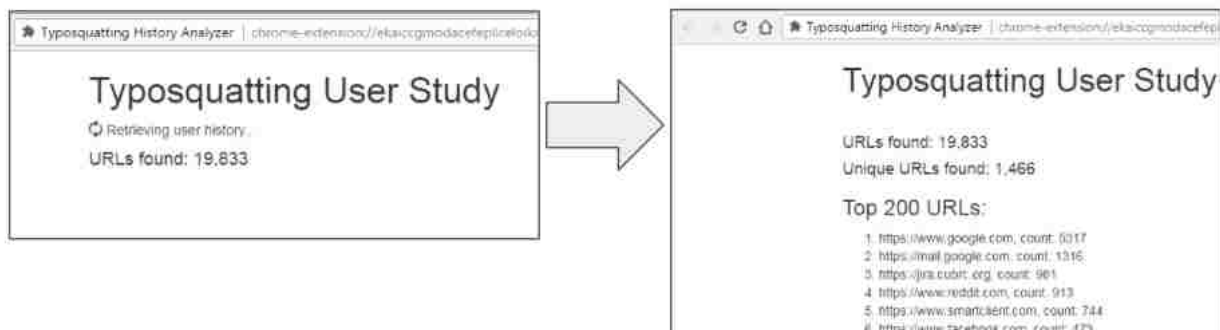


Figure 3.2: An example of a browser extension using the *chrome.history* API to extract the browser’s history.

Web browsing history, for example, is one of the most crucial types of information that will help

us achieve our goal. Being able to access a user’s browsing habits will allow us to generate specific malicious domain names that will be personalized for the target user. Essentially, these custom-crafted malicious domain names have a greater potential to “succeed” in deceiving the user. Accessing the browsing history of a web browser on a shared host may be problematic, as most modern browsers now have “private browsing” mode which does not store any history or cache between sessions. However, as recently shown by Zhao and Liu [125], browser extensions can circumvent this since they may not always abide by the private browsing policy; some keep data even after the private browsing session ends. For example, a Google Chrome browser extension was specifically developed to extract browsing history in an effort to profile a user. As shown in Figure 3.2, the use of the *chrome.history* API allowed us to access the browser’s database of visited URLs and query for them in the browser’s history.

### 3.2 User Study: Malicious Domain Names with Context

As a continuation of the user study presented in Section 2.3, we conducted a follow-up study to establish if taking context into account will result in a more “successful” attack strategy that utilizes malicious domain names. Similar to the previous study, subjects were first asked a series of demographic questions followed by a list of actual domain name URLs in which some were deliberately modified according to the typosquatting techniques shown in Section 2.2.1. The survey instructed subjects that they will be presented with several domain names that were sampled from Alexa’s most popular domain names and asked to indicate if they appear to be malicious or not by selecting “Yes” or “No”. Unlike our previous user study which sampled from the Alexa’s global list of top sites, this survey dynamically sampled from a pool of domain names that were generated based on a user’s input (*e.g.* country, subject interest, favorite websites). The primary objective here was to determine the effectiveness of “personalized” malicious domain names that takes the

context of the user into account.

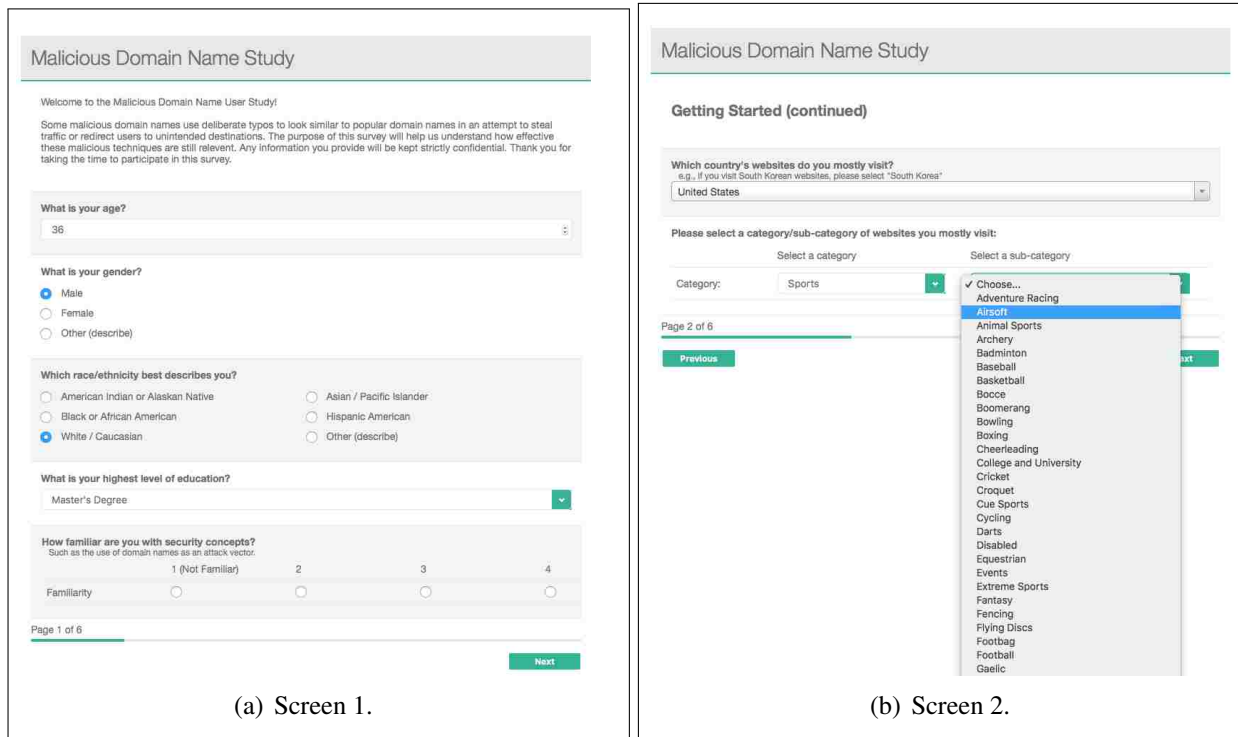


Figure 3.3: Screenshots of screens 1 & 2 of the context user study.

### 3.2.1 User Study Design & Implementation

Since we required the ability to dynamically sample domain names from the Amazon Web Services (AWS) API, we chose to develop and host our own custom web application. Rather than relying on third-party survey services like *Google Forms* or *SurveyMonkey*, we chose to adopt the newly released open-source *SurveyJS* library which allowed us to dynamically generate the survey on-the-fly. The survey web application was developed primarily using JavaScript, HTML, and PHP and featured a series of navigable pages with input questions with error-handling (*i.e.* blank or invalid responses). Following the introduction and consent page, Fig. 3.3(a) shows a series of

questions were presented to the user to gather demographical data: *Age, Gender, Race/Ethnicity, Education, and Familiarity of Security Concepts* (on a scale 1-5). Unlike the previous user study which asked the user to input the current time (necessary to calculate the total time spent), our custom-developed web application allowed us to easily record the actual time spent using JavaScript timers. Fig. 3.3(b) shows the next page where the user is prompted for which country's websites they mostly visit (*e.g.* if mostly South Korean websites, then select "South Korea"). They are also asked to select a pair of category/sub-category of websites they mostly visit, which is directly mapped from the categories specified by the AWS API (*e.g.* Computers/Security). The input from these questions are then used in web service API calls to AWS to return a list of top domain names for a particular country or category which will be included in the pool of available domain names discussed in the following section.

### 3.2.2 Selection of Domain Names

To avoid "survey fatigue" [17] and to ensure our subjects were not overwhelmed with answering several questions, we strove to limit the amount of domain names to 30. Rather than sampling from a pool of domain names representing the global top sites from Alexa, the pool of available candidate domain names was constructed dynamically using the Alexa top sites for specific countries and categories. For example, Alexa lists the third-most popular website in South Korea as `naver.com` (a.k.a. "the Google of South Korea") while its global rank is 110. If a subject chooses "Arts & Music" for a category/sub-category pair, then the top ranking results include websites such as `ultimate-guitar.com` and `billboard.com` which may be more familiar with that particular user.

To avoid the invasive approach of having an application attempt to scan your browsing history (as shown in Fig. 3.2), this study prompts the user for 5 of their most-frequently visited web-

sites (Fig. 3.4(a)). After validating each of the user-supplied domain names through the AWS API (to prevent invalid or non-existent domains), these 5 user-supplied domains along with 25 domains sampled from the country/category pool form a total of 30 candidate domain names to be presented to the user. Fig. 3.4(b) shows a “word cloud” visualization of all the user-supplied domain names with `google.com` and `naver.com` being the most-commonly supplied domains frequently visited by users.

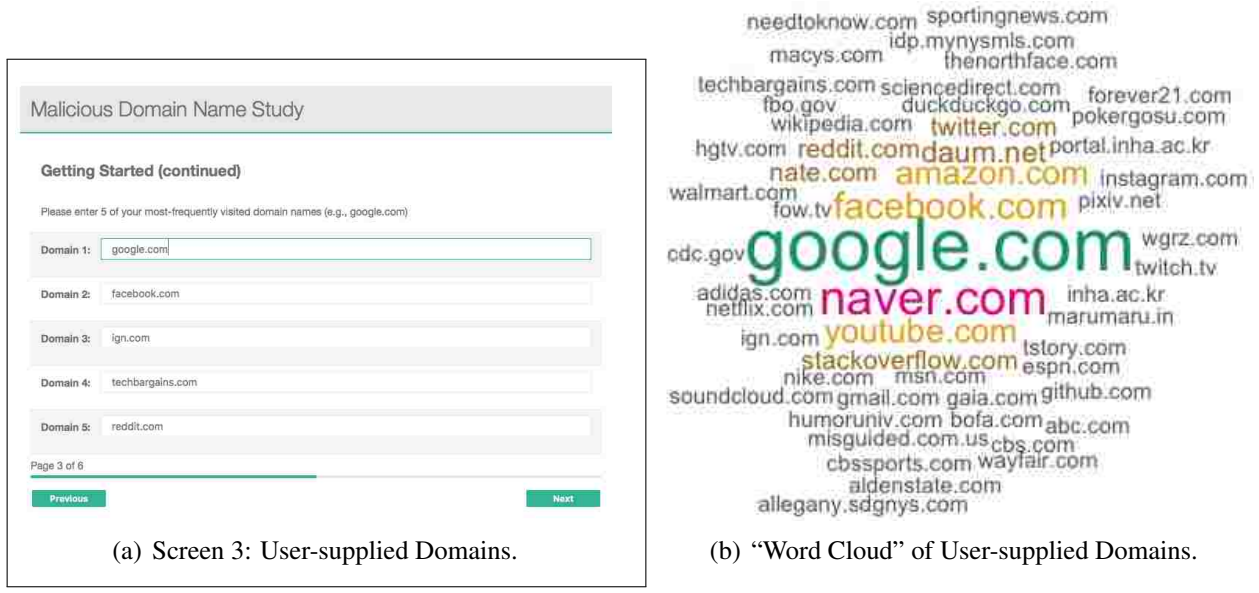


Figure 3.4: User-supplied domains from the context user study.

As before, we simply iterated through the candidate domains and randomly decided if it should be typosquatted or not with a random technique chosen from Section 2.2.1. For this study, we only considered *Models 2-7* in our study since *Model 1* (i.e. removing the ‘.’ in “www.”) was the most easily-recognizable typo technique that failed to deceive users. Fig. 3.5 shows a snippet of the algorithm that generates a typosquatted variant given a domain name and typo model number.

```

1 function generateTypo(input, model) {
2   if (model == 1) {
3     // "Missing-dot typos: the dot following 'www' is forgotten, e.g., wwwexample.com"
4     if (domain.indexOf('www.') !== -1) {
5       return domain.replace('www.', 'www');
6     }
7   } else if (model == 2) {
8     // "Character-omission: one character is omitted, e.g., www.exmple.com"
9     while (true) {
10      i = getRandomArbitrary(0, strLen);
11      if (domain[i] !== '.') {
12        if (i == 0) {
13          typoDomain = domain.slice(1);
14        } else {
15          typoDomain = domain.slice(0, i) + domain.slice(i + 1);
16        }
17        typoDomain = origPrefix + typoDomain + origSuffix;
18        if (isValidDomain(typoDomain)) {
19          return typoDomain;
20        }
21      }
22    }
23  } else if (model == 3) {
24    // "Character-permutation: two consecutive characters swapped e.g., www.examlpe.com"
25    var sample = range(strLen - 1);
26    shuffle(sample);
27    for (i = 0; i < sample.length; i++) {
28      var next = i + 1;
29      if (domain[i] !== domain[next]) {
30        typoDomain = domain.slice(0, i)
31        typoDomain += domain[next] + domain[i] + domain.slice(next + 1);
32        typoDomain = origPrefix + typoDomain + origSuffix;
33        if (isValidDomain(typoDomain)) {
34          return typoDomain;

```

Figure 3.5: JavaScript code snippet for generating a typosquatted domain name.

### 3.3 User Study Results

Our preliminary study included 18 participants who completed the survey with a median time of 410 seconds (approx. 7 minutes). Similar to our first study, our evaluation primarily focused on two performance metrics:

- (i) Correct responses
- (ii) Amount of time to complete the survey

As before, a correct response is defined as to whether the user answered “No” if the given domain name was an authoritative domain name (*unaltered*) or “Yes” if the given domain name was altered

according to Section 2.2.1. As illustrated in Fig. 3.6(a), the scores ranged from 18 to 29 (out of 30) with a mean, standard deviation and variance of 22.5, 2.98, and 8.85, respectively. The completion time ranged from 125 seconds (approx. 2 minutes) to 1,630 seconds (approx. 27 minutes) with (Mean=544.4, s.d.=358.1, variance=128,268.3). The longest completion time may have been attributed to the fact that a survey participant may have left the survey website and returned at a later time to complete it. In future work, we will ensure a timeout mechanism is built-in to warn the user that a timeout will occur if no activity is detected after some threshold.

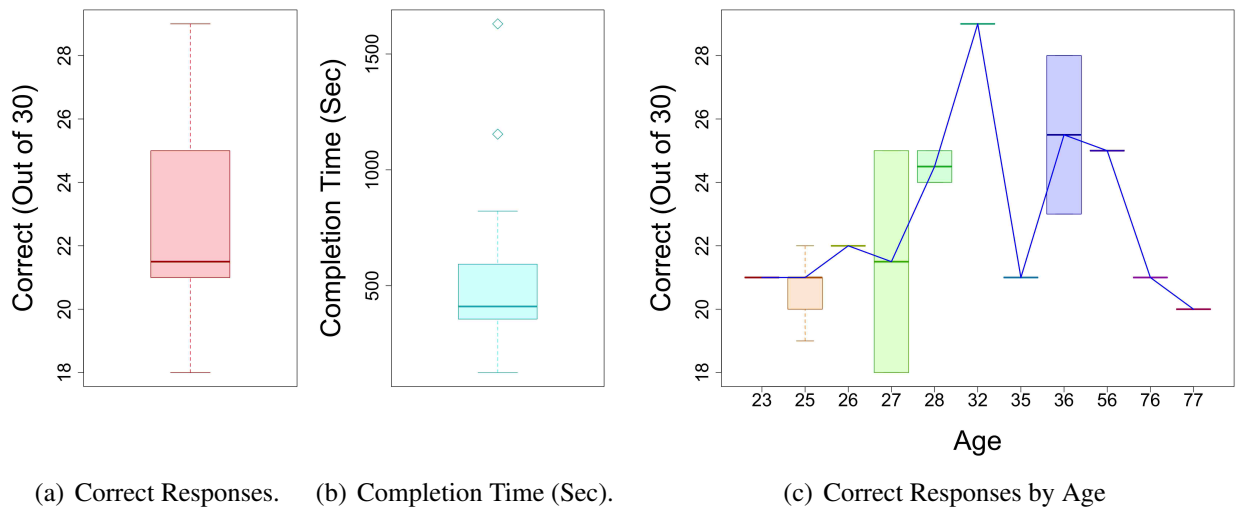


Figure 3.6: Overall correct responses and completion time, as well as correct responses by age.

### 3.3.1 Demographics

The ages of the participants ranged from 23 to 77 (Mean=35.1, s.d.=16.9, variance=285.2). As shown in Fig. 3.6(c), there is an upward trend in the number of correct responses followed by an overall decline as the age of the participants increased. This is in contrast to the results of our first user study shown in Fig. 2.4(a) where those younger participants scored higher, but it should be



noted that the range of ages was only from 22 to 39. Also in contrast to the first user study was the results of the correct responses grouped by a participant’s education, which shows an upward trend as the higher educational degree increases (Fig. 3.7(a)). Ethnicity was a new demographic question introduced in this user study, which shows that participants who identified themselves as “White / Caucasian” (who are most likely native English speakers) as having the highest scores among the ethnic groups (Fig. 3.7(b)).

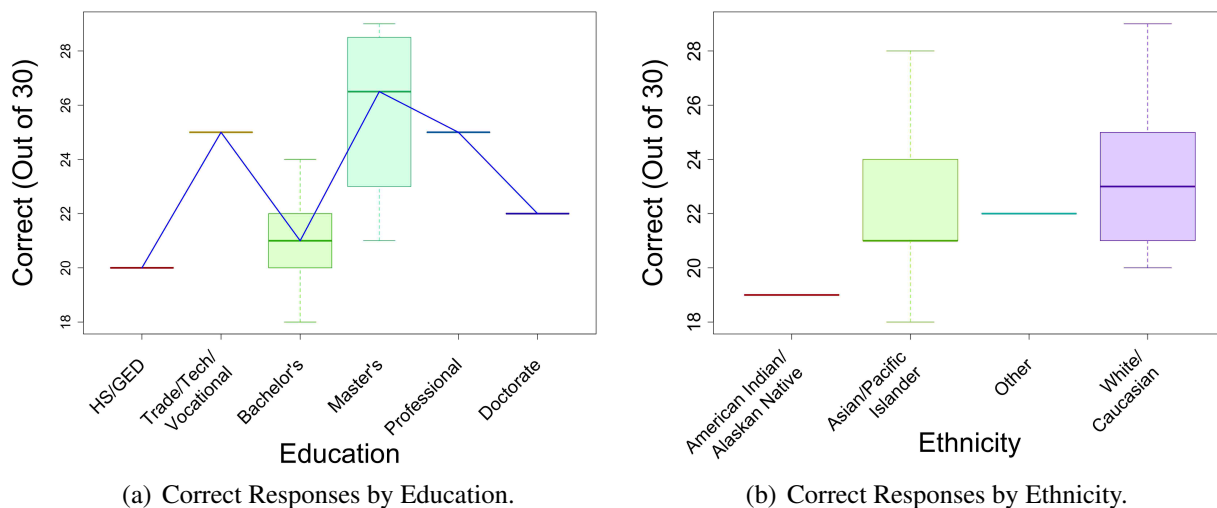


Figure 3.7: Plots of the context user study according to education and ethnicity.

### 3.3.2 Typosquatting Model Effectiveness

Similar to our findings from the first user study, the typo model which caused the most participants to incorrectly identify typosquatted domain names was *Model 2* (Character-omission). As can be seen in Fig. 3.8(a), *Model 2* had the least amount of correct identifications and most amount of incorrect identifications of typosquatted domain names. Table 3.1 also reinforces the previous fact, which depicts the top incorrectly identified domains which were supplied by the participants them-

selves. It should be noted that the top domains in Table 3.1 were from South Korean participants, as the first entry (`tstory.com`) is most likely a misspelling of `tistory.com` (a popular South Korean blog-publishing service).

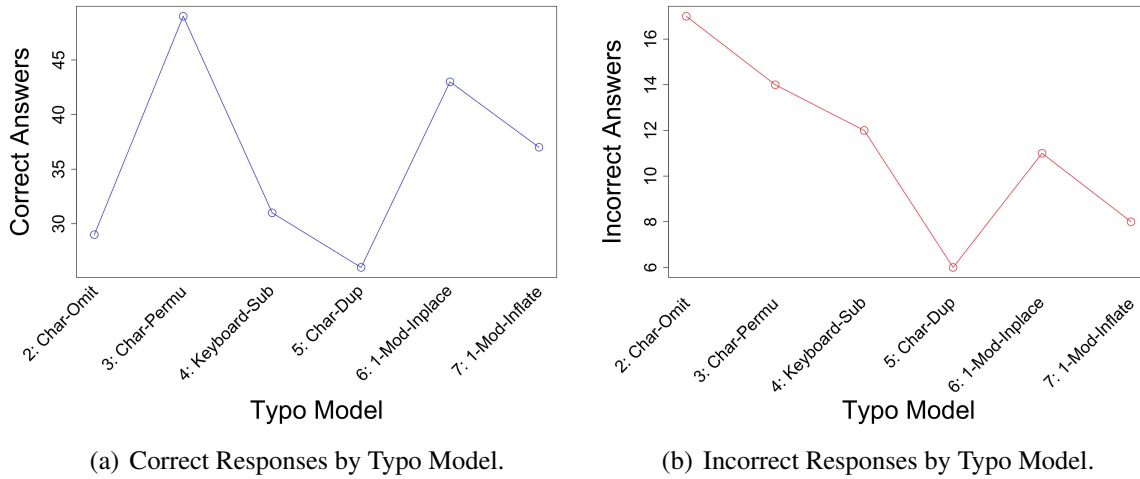


Figure 3.8: Plots of the context user study according to correct/incorrect by typo model.

Table 3.1: Top incorrectly identified user-supplied domains (unmodified shown in gray).

Domain	Orig. Domain	Typo Model	User Specified?	Num Correct	Num Incorrect
tstory.com	---	---	Yes	0	1
naver.com	---	---	Yes	4	1
stackoverlow.com	stackoverflow.com	2 (char-omission)	Yes	0	1
gogle.com	google.com	2 (char-omission)	Yes	1	1
bofa.com	---	---	Yes	0	1
miguidded.com.us	misguided.com.us	2 (char-omission)	Yes	0	1
amanzon.com	amazon.com	7 (1-mod-inflate)	Yes	0	1
dam.net	daum.net	2 (char-omission)	Yes	0	1
fb6o.gov	fbo.gov	7 (1-mod-inflate)	Yes	0	1

### 3.4 Conclusion

Due to the lack of context, certain malicious domain names are more successful than others in deceiving users. The direction of this work explores the possibility of taking context into account when devising attack strategies that utilize malicious domains. One such method is profiling users by co-locating on the same host in an effort to extract vital information, such as browsing history, to aide in the development of advanced attacks. Such data exfiltration is possible with a browser extension that can programmatically extract the user’s browsing history.

In avoiding the previous invasive approach (and possible privacy concerns), we opted to conduct another online user study to determine if context can play a role in generating successful malicious domain names. To establish context, the online survey asked participants which country’s websites they most-frequently visit in addition to a specific category of websites (*e.g.* Arts & Music). Their responses were then used as inputs into web service calls to the Amazon Web Services (AWS) API for Alexa domain name ranking information, which returns the top domains grouped by countries and categories. We also prompted participants to enter 5 websites they visit frequently, which is then mixed into the domain pool and fed into an algorithm that applies a random typosquatting model. To account for generating dynamic survey questions, a custom web application was developed using open-source technology such as the *SurveyJS* JavaScript library.

The results of this study confirms that the most “successful” typosquatting technique for an adversary to utilize is *Model 2* (character-omission), which should be applied to domain names which takes the context of a user into account (*e.g.* the country and categories of websites they visit). It should be noted that in order for this to be “successful” in deceiving users, the character that is omitted from the domain name should *not* be the first and last characters. This is due to the so-called “jumbled word effect” that was discussed in Section 2.5.4, which is how our brains perceive the placement of characters in words. This phenomenon may help explain that if a user

sees a domain name they are very familiar with (*i.e.* context-specific domain names such as `stackoverflow.com` from Table 3.1), which also happens to be typosquatted with a character missing in the middle (*e.g.* `stackoverlow.com`; missing character ‘f’), then there is a high probably they will perceive the typosquatted domain name as the authoritative domain name they are familiar with and thus be deceived by our adversary.

## CHAPTER 4: PROACTIVE DETECTION OF ALGORITHMICALLY GENERATED DOMAIN NAMES<sup>1</sup>

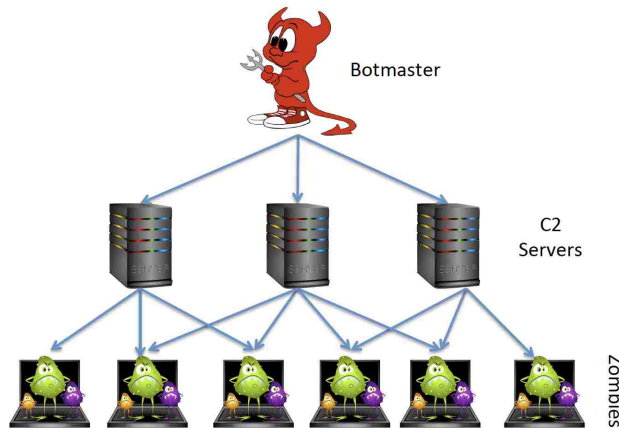


Figure 4.1: An illustration of a botnet infrastructure.

In this chapter, we examine another type of malicious domain name that is primarily used for command & control (C2) communications within botnets—which are becoming increasingly one of the most prevalent threats on the Internet [37, 111, 113, 112]. The typical botnet consists of various infected hosts, command and control (C2) channels, and a botmaster. The infected hosts (“zombies”), as shown in Figure 4.1, are often massively distributed, whereas the command and control is a channel used by a mastermind (the “botmaster”) to instruct bots to perform various forms of malice; *e.g.* launching DDoS attacks [114]. To communicate with bots, there are several potential ways utilized by botmasters, and domain names as a C2 channel are one of the most common and preferred methods—because they are easy to acquire and recycle [69]. To generate such domain names, domain generation algorithms (or DGAs) are widely used today by botmasters. Usually,

---

<sup>1</sup>This content was reproduced from the following article: J. Spaulding, J. Park, J. Kim, and A. Mohaisen, “Proactive detection of algorithmically generated malicious domains,” *In 2018 International Conference on Information Networking*, (ICOIN), pages 2124, Jan 2018. The copyright form for this article is included in the appendix.

DGAs use time as a seed to dynamically and automatically generate potentially pseudorandom domain names that are registered by botmasters and used by bots for C2 communication. Variants of the Mirai botnet, for instance, started to adopt DGAs to better avoid detection and keep a constant contact with C2 servers [30]. One way of mitigating these botnets is to prevent them from registering their C2 domains in the first place, or by taking such domain names down [89, 77].

**Detecting DGA-based Domains.** To address DGAs by detection, there have been two schools of thought either: 1) relying on reverse engineering of the bot software [85] or 2) using the intrinsic features of the generated domains [119]. The first method, while powerful in generating all domain names to be potentially used by a malware family (even in the future and can thus be used to proactively block those domains by pre-registering them), is very expensive. This method requires obtaining samples of the malware family that utilizes such DGAs. However, obtaining such malware is not the biggest hurdle: many of today's malware families employ obfuscation techniques that make their analysis a difficult task.

The second school of thought uses pseudorandomness of algorithmically-generated domains and exploits the fact that those domains have a high entropy for their detection [29]. Registered domain names that are queried by infected hosts are evaluated, a measure of their pseudorandomness using their entropy is calculated, and the likelihood of them being malicious based on their entropy score is assigned. While shown to identify malicious domains reasonably well, such techniques suffer from various drawbacks. First, domain names need to be registered in order for a monitor to be able measure such entropy and determine if a domain is malicious or not. Thus, such techniques cannot be used proactively to detect malicious domains. Second, those techniques assume that randomly generated domain names are only used in malicious activities: it is not far-fetched to imagine that domain names with high entropy are utilized for domain name parking, non-public facing domains (*e.g.* content delivery network (CDN) addressing), among others.

## 4.1 Motivation

To this end, this work addresses the problem of proactive malicious domain name identification focusing on DGAs. We aim to identify such domain names before they are registered using our detection system known as DRIFT. Our main source of inference is the DNS query and resolution of domain names. We motivated for our study by a large-scale analysis of domain names and their queries. We find that, domains that are used for malicious activities, and especially those generated algorithmically, tend to have unique and distinguishing patterns. In particular, domain names that are algorithmically generated tend to have a large number of DNS queries even before their registration, typically resulting in NXDomain (non-existent domain) responses. This trend persists, and the number of queries increases and peaks at the time of registration, then declines gradually—indicating the ephemeral use of those domain names for their major purpose. On the other hand, domain names that are being used for benign applications tend to have significantly less queries before registration, while their post-registration query volumes (which may fluctuate over time) do not have a single declining curve, thus highlighting a fundamentally different use model.

**Contributions.** The contribution of this work is as follows. First, we highlight a fundamental difference between the query patterns for domain names that are used by botnets, often generated using DGAs, and those by benign ones. We use this insight to differentiate between those domain names using a simple classification algorithm for proactive detection of malicious domains.

## 4.2 Related Work

### 4.2.1 Domain Generation Algorithm

Primarily used for a botnet's C2 channel, DGAs are widely used in malware families such as Conficker, Kraken and Torpig. As the potential impact of DGAs continues to expand, a wide range of research has been well underway in the broad field of computer security. One of the first academic publications that introduced the concept of malware utilizing DGAs was in 2009 by Stone-Gross *et al.* [109], who described their experiences attempting to seize control of the Torpig botnet. They pointed out that in the past, botnet authors would use IP fast-flux techniques where a certain domain is mapped to a set of frequently changing IP addresses to avoid take-downs. Realizing that a single domain name constituted a single point failure, the authors discovered that the Torpig botnet starting using a *domain flux* technique that employed the use of a DGA for locating its C2 server. Shin *et al.* [102, 103] conducted a large-scale survey of the distribution of Conficker (one the most notorious DGA-based malware) and the effectiveness of existing detection systems. The Kraken botnet, another notorious DGA-based malware, was closely analyzed by Royal [98] who detailed its behavior and provided sample DGA domains as well as MD5 hashes. Recently, Fu *et al.* [50] proposed a pair of DGAs that can avoid the latest detection methods by using hidden Markov models (HMMs) and probabilistic context-free grammars (PCFGs).

### 4.2.2 Detection of DGA-based Botnet

To counter the prevalence of DGA-based malware, several approaches have been proposed over the years to identify algorithmically-generated malicious domain names by DGA-based botnets. As mentioned previously, all of these works usually fall into two schools of thought either: 1) relying on reverse-engineering of the bot software or 2) using the intrinsic features of the generated



domains. Table 4.1 summarizes the most recent studies on how to proceed with the detection of DGA-based botnets. As a result of the development of machine learning technology, the rate of research for identifying algorithmically-generated domains using various intrinsic features is high.

Table 4.1: Summary of DGA Domain Identification Approaches.

Study	Approach	Malware Families
Stone-Gross <i>et al.</i> 2009 [109]	1) Reverse-engineer	1 (Torpig)
Yadav <i>et al.</i> 2012 [120]	2) Intrinsic	3 (Conficker, Torpig & Kraken)
Antonakakis <i>et al.</i> 2012 [29]	2) Intrinsic	12 Botnets
Barabosch <i>et al.</i> 2012 [32]	1) Reverse-engineer	4 Families
Guerid <i>et al.</i> 2013 [55]	2) Intrinsic	2 (Conficker & Kraken)
Zhang <i>et al.</i> 2013 [124]	2) Intrinsic	3 (Conficker, Torpig & Srizbis)
Haddadi & Zincir-Heywood 2013 [56]	2) Intrinsic	3 (Conficker, Kraken & Alexa)
Davuth & Kim 2013 [46]	2) Intrinsic	3 (Conficker, Torpig & Kraken)
Mowbray & Hagen 2014 [88]	2) Intrinsic	19 DGAs
Schiavoni <i>et al.</i> 2014 [100]	2) Intrinsic	3 (Conficker, Torpig & Bamital)
Bilge <i>et al.</i> 2014 [35]	2) Intrinsic	2 (Conficker, Torpig)
Sharifnya & Abadi 2015 [101]	2) Intrinsic	3 (Conficker, Kraken & Murofet)
Grill <i>et al.</i> 2015 [54]	2) Intrinsic	6 families
Wang <i>et al.</i> 2016 [116]	2) Intrinsic	1 (Conficker)
Kwon <i>et al.</i> 2016 [72]	2) Intrinsic	26 Botnets
Zhang <i>et al.</i> 2016 [122]	2) Intrinsic	2 (Conficker & Kraken)
Plohmann <i>et al.</i> 2016 [94]	1) Reverse-engineer	43 Families
Wang <i>et al.</i> 2016 [115]	2) Intrinsic	3 (newGoZ, Ramnit & Qakbot)

**Approach through reverse-engineering.** The recent work by Plohmann *et al.* [94] is the epitome of the first school of thought that relies on reverse-engineering malware. The authors performed an all-encompassing study of several malware families featuring DGAs in a bottom-up fashion by reimplementing their algorithms to ultimately produce over 159 million unique DGA domains. Using a WHOIS dataset from DomainTools (containing over 9 billion WHOIS records spanning

14 years) the authors removed any DGA domain for families not in the WHOIS dataset, which left them with a set of over 18 million unique DGA domains. By studying the registration status of these DGA domains, they were able to determine that their dataset can be employed as a black-list for C2 domains in addition to the identification of malware families with virtually no false positives.

**Approach employing intrinsic features.** More research is being conducted to identify malicious domains through intrinsic features such as behavior and dynamics in DGA-based botnets. Yadav *et al.* [120] conducted a study that computed the Shannon entropy for the distribution of n-grams for groups of domain names. Antonakakis proposed *Pleiades* which focuses on NXDomain responses. *Pleiades* classifies the domains based on the similarity to detect malicious domain names based on known models of DGA. The approach suggested by Guerid *et al.* [55] performs bot grouping based on NXDomain responses in privacy-preserving manner by using Bloom filters. Zhang *et al.* [124] built the system detecting the DGA-based malicious domain names by taking entropy, bigram, and length into account. Haddadi & Zincir-Heywood [56] proposed a Stateful-SBB classifier based on the genetic programming (GP) which takes string domain name as input to determine whether it is malicious. Davuth & Kim [46] implemented a Support Vector Machine (SVM) classifier which only takes a domain name as input to reduce the burden of collecting and managing large amounts of metadata. Mowbray & Hagen [88] identified DGA domain names by analyzing client IP addresses with abnormal distributions of second-level strings lengths in their DNS queries. *Phoenix* presented by Schiavoni *et al.* [100] labels the malicious domain names automatically using DNS and IP-related features, which can be applied to not only groups of domains but also a single domain. Bilge *et al.* [35] proposed a system, *Exposure*, that utilizes 4 time-based, 4 DNS answer-based, 5 TTL value-based, and 2 domain name-based features to detect DGA-based domains. *DFBotKiller* proposed by Sharifnya & Abadi [101] is a reputation-based system which distinguishes malicious domains by considering suspicious activities as well as DNS

query failures. Grill *et al.* [54] presented a DGA-malware detection system only using NetFlow information, which is a gathering of identical protocol packets representing the communication between the source IP/port to a destination IP/port pair. Wang *et al.* [116] employs social network analysis which divides the hosts and NXDomains into clusters and identifies them as either benign or malicious. *PsyBoG* proposed by Kwon *et al.* [72] is a malicious domain name detection system that uses a signal processing technique and power spectral density (PSD) analysis. Zhang *et al.* [122] presented *Botdigger* which looks at data from single network to identify individual bots. Wang *et al.* [115] devised *BotMeter* which assesses the populations of DGA-based botnets over large networks by analyzing the DNS lookups at upper-level DNS servers.

### 4.3 Preliminaries

Section 3.1.1 provided an overview of how the DNS functions and the domain name resolution process once a user initiates a DNS query. For this section, we discuss the procedure of how a domain name is registered in the DNS and how DNS queries can be skewed by “clock drift”.

#### 4.3.1 Domain Name Registration Process

In late 1998, the United States Department of Commerce named the newly-formed non-profit ICANN (Internet Corporation for Assigned Names and Numbers) as the new entity to oversee the assignment of both IP addresses and domain names [13]. As outlined by ICANN, the process works as follows [20]: a domain name Registrant (a person or organization) will submit an online application to an ICANN-accredited domain Registrar (*e.g.* *GoDaddy*) or through a third-party Reseller. The Registrar will in turn generate a WHOIS entry populated with the applicant’s data, after determining if the domain name is not already registered. Note that while these Registrars

manage the daily transactions of selling domain registrations, it is the Registries (*e.g.* Verisign) that are primarily responsible for a TLD’s registry. These registries are also responsible for accepting registration requests, and as ICANN states [20]: “maintaining a database of the necessary domain name registration data and providing name servers to publish the zone file data (*i.e.* information about the location of a domain name) throughout the Internet.”

### 4.3.2 Understanding Clock Drift

Since our approach exploits the fact that DGA domains tend to receive a large number of DNS queries before their registration, we attribute this to a phenomenon known as “clock drift”. A drift is introduced by many factors, as Jackson [62] states: “...including network jitter, delays introduced by software, and even the environmental conditions in which the computer is operating”. In the following, we discuss the history of how hardware clocks were introduced into PCs and what could contribute to their drift.

**How a Computer Keeps Time.** Back in late 1981 when the original IBM PC was introduced, most first-generation PCs did not include real time clock (RTC) hardware. As Becker [33] mentions, “whenever the system was booted up, the date and time was initialized to 1980/01/01 00:00:00 and the user was expected to manually set them”. With the introduction of the IBM-AT in 1984, all PC-compatible computers have now kept time the same way: software and hardware clocks [80]. The software clock runs when the computer boots up and stops when the computer shuts down.

The hardware clock, on the other hand, uses a battery and runs even while the computer is turned off. The hardware clock uses a crystal oscillator that runs at a specific frequency of 32.768 kHz, which is described by [104] as “the magical number for computer-based clocks because it is easily divisible into 1Hz by counting every  $2^{15}$  pulse”. However, it has been shown that computer

manufacturers will tend to use inexpensive crystals which affect the clock's accuracy [62].

**Drifting apart.** As Kohno *et al.* point out [68], the clocks on machines that are managed professionally by administrators tend to be synchronized with the correct time via Network Time Protocol (NTP). Non-professionally managed machines, on the other hand, are less likely to be synchronized by an external time source. For example, the default behavior on recent versions of the Windows operating system is to synchronize the system time with Internet servers on a weekly basis [5].

#### 4.4 Dataset and Characteristics

Our dataset was originally used by Thomas and Mohaisen in their work on detecting and clustering botnet domains using DNS traffic [111]. The dataset consists of DNS traffic from July 2012 from a large DNS operator's authoritative name servers for the COM, NET, TV and CC top-level domains (TLDs). As the registry of large TLDs, this large DNS operator has a global view of DNS traffic, giving a unique observation of malware-associated DNS traffic.

##### 4.4.1 Malware Data

Conficker is one of the most well-known malware samples that employed the use of DGAs [86]. The family was originally discovered in 2008, and has been active for the past several years by infecting many hosts world-wide and by mutating several times (*i.e.* variants A through E [102]). Through reverse-engineering, the coalition of public and private researchers known as the Conficker Working Group [11] was able to successfully determine the domain names that are generated daily for multiple variants of Conficker. As shown in Table 4.2, the Conficker variants A and B generate hundreds of DGA domains daily while variant C generates 50K domains per day

across several TLDs. Since the A and B Conficker variants generate domains in the COM, NET and CC TLDs, we can conduct our analysis using the DNS traffic dataset mentioned earlier. By mid-April 2009 all of the domain names to be generated by Conficker A were registered, thereby severing further update attempt by the malware [82]. With Conficker variants A and B generating a combined total of 15,500 domains in July 2012, [111] showed that 30 of the DGA domains were registered in either the COM or NET TLD with their name servers returning YXDomain (name exists when it should not) responses. For the rest of the generated DGA domains, much of traffic resulted in NXDomain (non-existent domain) responses.

Table 4.2: Conficker DGA by variant and domains per day. Adapted from [111].

Variant	Domains / Day	TLDs
A	250	biz, info, org, net, and com
B	250	biz, info, org, net, com, ..., cn
C	50K	110 ccTLDs not tv or cc

#### 4.4.2 NXDomain Data

When a domain name is not registered, any queries for that particular domain name will not resolve to an IP address and will return the NXDomain response. This term was originally used to represent DNS response code 3 in RFC 1035 [84] and RFC 2308 [27]. As mentioned above, the following data represents the state of the DNS traffic gathered from a large DNS operator in July 2012. We emphasize that only the TLDs (*e.g.* COM, NET, TV and CC) for which we had DNS traffic available for that fell with the TLDs of the Conficker DGA domains (shown in 4.2) were examined.

In our dataset, as described by Thomas and Mohaisen [111], “a typical day in the COM zone has 2.5 billion NXDomain requests for more than 350 million unique second-level domains while NET receives around 500 million NXDomain requests for more than 60 million unique second-

level domains”. Since the TV and CC TLDs have considerably less domains than COM and NET, they naturally receive far less NXDomain requests. We also noticed that for a given epoch of time, the bulk of specific NXDomains that were observed only received a few requests compared to the large volumes of daily requests to specific domain names. Figure 4.2 (subplot for each zone) shows the cumulative distribution of NXDomain traffic received daily, which demonstrates that over 95% of specific 2nd-level NXDomains receive fewer than 10 requests daily.

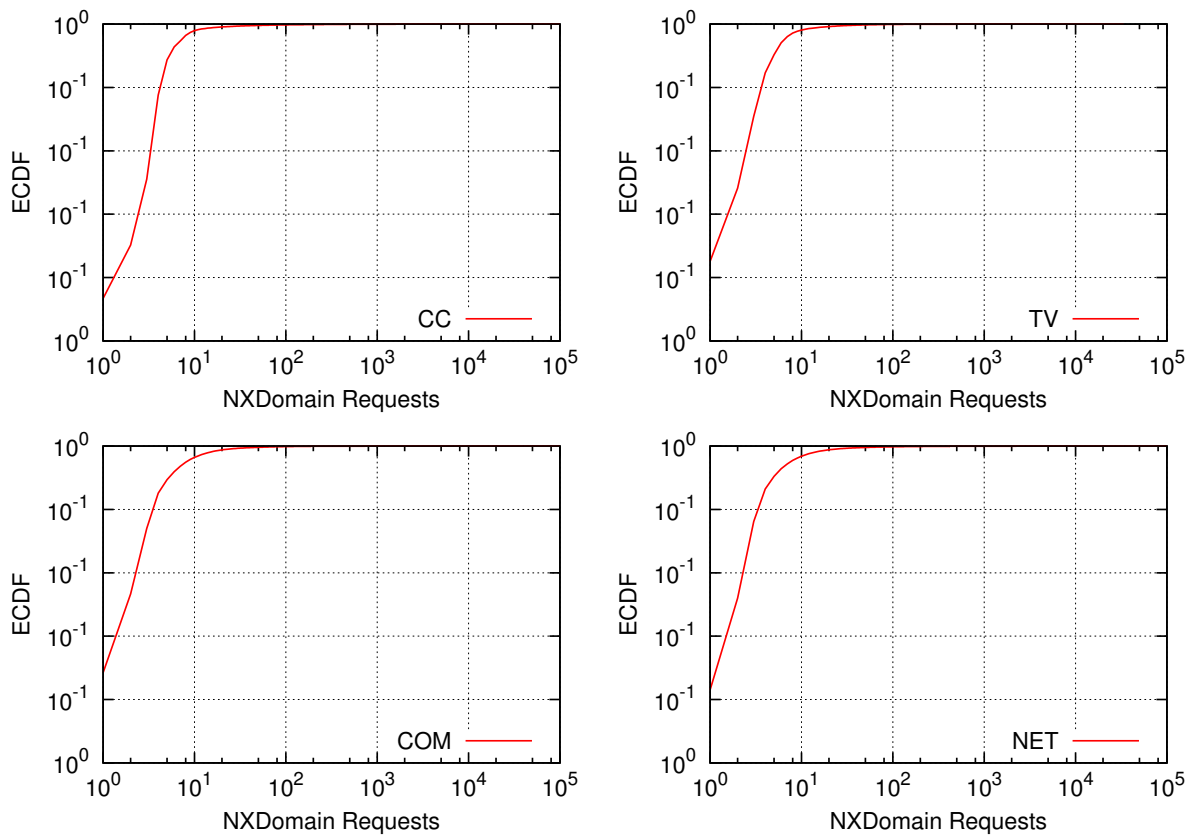


Figure 4.2: NXDomain traffic volumes to major TLDs: .cc, .tv, .com, and .net, respectively. Notice that the majority of domains receive small number of queries, and a small percentage ( $\sim 3\%$ ) receive more than 10 queries.

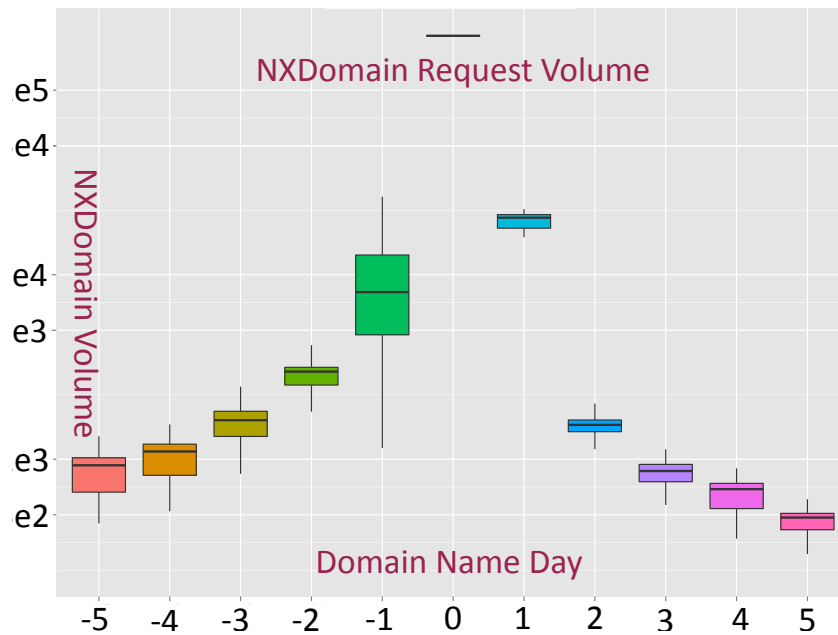


Figure 4.3: Conficker NXDomain DNS lookups.

#### 4.4.3 Conficker NXDomain DNS Traffic

The bulk of DGA domains generated by Conficker is classified under the NXDomain category. To grasp the creation process of a DGA domain, we examined various aspects of the DNS traffic before, during and after the domain’s generation date. Utilizing the DGA domains produced by the Conficker Working Group [11] mentioned previously, we grouped them according to generation date and examined their DNS traffic. Similar to the approach by [111], “for a given domain to be generated on day  $x$ , we measured the domain’s DNS traffic on days  $x - 5$  to  $x + 5$ ”. As shown in Figure 4.3, the box plots depict the DNS traffic patterns 5 days before and after the generation of a DGA domain (day 0) from Conficker variant B. No matter the generation date, this plot clearly shows that DGA-based domains see considerable amounts of DNS queries before and after they are generated by the malware.



## 4.5 Online Detection Algorithm

Compared to the whole population of NXDomain traffic across several TLDs, the specific NXDomain traffic generated by DGA-based domain names makes them stand out. We saw previously that this NXDomain traffic spikes both the day before and after the domain is generated by the DGA. As noted by [111], for an average DGA domain: “traffic volumes on the exact generation date soar several magnitudes higher than the  $\pm 5$  days baseline to 42,887 unique /24 recursive name servers consisting of 199,097 total NXDomain requests from 211 unique countries”. In the following, we describe the rationale behind our detection algorithm and walk through the models created for the classification process.

### 4.5.1 Rationale of Feature

There are potentially various explanations for why domain names get queried before their registration. First, domain names intended for benign usage might get queried by interested registrants who want to acquire them, thus explaining the small number of queries some of them receive before registration. On the other hand, the large number of queries that DGA domains receive might have to do with the fact that those domain names are time-dependent. As highlighted in Figure 4.4 for the DGA domain used initially by CryptoLocker, a bot calculates a domain name using time features as the main inputs. A large drift in the time or clock misconfiguration would result in bots contacting the domain name before or after its registration.

We use such verified observation as the main feature for identifying malicious domain names. Using this *a priori* knowledge captured in a training model, we devise a method that can detect those domains even before they are registered with a high accuracy rate. The proposed approach based on this feature has various plausible benefits over the state-of-the-art. Among others, the

proposed technique is robust to the behavior of the adversary.

```
1 def generate_domain(y, m, d):
2     domain = ""
3     for i in range(16):
4         y=((y^8*y)>>11)^((y&0xFFFFFFFF)<<17)
5         m=((m^4*m)>>25)^16*(m&0xFFFFFFFF8)
6         d=((d^(d<<13))>>19)^((d&0xFFFFF8)<<12)
7         dm+=chr(((y^m^d)%25)+97)
8     return domain
```

Figure 4.4: An example of the algorithm used by CryptoLocker for initially generating algorithmic domain names for command and control [78].

#### 4.5.2 Online Detection

In determining the maliciousness of a given domain name, we use the notion of the difference function. Given a function  $y = f(x)$  that is defined on an interval  $[x, x + h]$ , the average rate of change of the function on the interval  $[x, x + h]$  is:

$$\frac{f(x+h) - f(x)}{(x+h) - x} = \frac{f(x+h) - f(x)}{h} \quad (4.1)$$

For an interval of  $[x - a, x + a]$ , ( $a$  is some constant), we have:

$$\frac{f(x+a) - f(x-a)}{(x+a) - (x-a)} = \frac{f(x+a) - f(x-a)}{2a} \quad (4.2)$$

Using this simple concept, in the following we outline how to build a feature vector, how to build a model, and how to label various domain names as malicious (used for C2) or benign.

#### 4.5.2.1 Building Feature Vectors

For a window of size  $2a$  ( $a$  from left and right of a point  $x$ ; note that  $x$  here can be any point in time), we calculate the difference as the change in the number of NXDomain responses to a given domain, normalized by the total time units corresponding to  $2a$ . The parameter  $a$  is used based on the desired performance, and  $x$  is used for all values of the observed traffic. We highlight the operation of the basic feature with an example. Let's consider an observation of  $[o_1^j, o_2^j, \dots, o_k^j]$  (for a domain  $j$ ), where each observation is the total number of NXDomain responses for a domain  $j$  over a fixed period of time (e.g. hour). If we are to consider  $a = 1$  in Eq. 4.2, we calculate  $f_i^j$  as  $|o_{i+1}^j - o_i^j|/2$  for all  $i$  (resulting in a vector of values, representing the use of the given domain;  $[f_i^j]^{1 \times k}$ ). For a unified treatment of the vector, we normalize each element in it by the sum of all of its elements; this is  $f_i^j / \sum_{\forall i} f_i^j$ . Our detection algorithm then uses the same idea as above, over a sliding window of observations. As time goes, the window slides by forgetting the oldest observations of NXDomain responses for the given domain. Additionally, the detector updates the count vector of the NXDomain responses for the domain, and calculates our feature vector as the difference function.

#### 4.5.2.2 Building a Model

Given a set of malicious domain names  $d_1, \dots, d_t$ , we create model  $\mathcal{M}$  that is calculated as a centroid feature vector corresponding to the average of the feature values of the different domains. As such, we define  $\mathcal{M}$  as:

$$\mathcal{M} = [m_1, \dots, m_k] : m_i = 1/t \sum_{j=1}^t f_i^j \quad (4.3)$$

As above, for a unified treatment, we normalize each element in it by the sum of all of its elements; this is  $m_i / \sum_{\forall i} m_i$

#### 4.5.2.3 Learning Labels of Domains

The labeling of the domains is divided into two learning types: 1-class, which is concerned with the associated label if the distance between the feature vector and a reference is at most  $\Delta$ , and 2-class learning, which is concerned of associating a domain name with the label that is closest to it (based on the comparison of the feature vector corresponding to each of them). Below, we will describe both approaches formally.

**1-class learning.** For a candidate domain  $x$  defined by its difference function  $f^x$  as above, we determine the label of the domain by conducting the following. We calculate the Manhattan distance between  $\mathcal{M}$  and  $f^x$ . That is, we calculate:

$$D(\mathcal{M}, f^x) = \sum_{\forall i} |m_i - f_i^x| \quad (4.4)$$

Then, we label the domain as malicious if  $D(\mathcal{M}, f^x) > \Delta$  and as benign otherwise.  $\Delta$  is determined through measurements and tuning, based on the learning of the underlying distribution of the NXDomain queries and their difference functions of malicious domains.

**2-class learning.** Alternatively, we create a model for a set of known benign domains, namely  $\mathcal{B}$ , where  $\mathcal{B} = [b_1, \dots, b_k]$  and assign the label of a sample  $x$  based on the following:

$$\text{Label} = \begin{cases} \text{Malicious} & : D(\mathcal{B}, f^x) > D(\mathcal{M}, f^x) \\ \text{Benign} & : D(\mathcal{B}, f^x) \leq D(\mathcal{M}, f^x) \end{cases} \quad (4.5)$$

We note that our scheme is less aggressive, since it prioritizes benign over malicious, as shown above. Depending on the detector objective, we might also be more aggressive by assigning a malicious label to a domains when the two quantities are equal; this is, alternatively:

$$\text{Label} = \begin{cases} \text{Malicious} & : D(\mathcal{B}, f^x) \geq D(\mathcal{M}, f^x) \\ \text{Benign} & : D(\mathcal{B}, f^x) < D(\mathcal{M}, f^x) \end{cases} \quad (4.6)$$

### 4.5.3 Detection Algorithm

The approach behind DRIFT in classifying domains into “malicious” or “benign” is based on the supervised learning technique of the Nearest Centroid Neighborhood (NCN) classifier introduced by Chaudhuri in 1994 [38]. The key idea behind this algorithm is that it assumes a target class is represented by a cluster and uses its mean (*i.e.* centroid) to determine the class of a new sample point based on its distance. Typically, Euclidean distance calculations are used, but this can be any distance function. In DRIFT, we calculate the Manhattan distance between feature vectors as shown in Equation 4.4. For a sample with an unknown class, the NCN classifier chooses a class with the closest centroid for the given sample. Despite this simple approach, Chaudhuri emphasizes that the NCN classifier can obtain high accuracy [38]. As shown in a study by Han and Karypis [57], “the centroid-based classifier consistently and substantially outperforms other algorithms such as Naive Bayesian, k-nearest neighbors (k-NN), and C4.5, on a wide range of datasets”. The k-NN algorithm, for example, requires computing and sorting every distance between the unknown sample and all others in the dataset--which is computationally expensive when the dataset is very large.

The first step in DRIFT is to build a model based on a set of confirmed malicious domain names, which is outlined in section 4.5.2.2. Using a set of NXDomain observations for each malicious do-

main, the feature vectors are built using the function shown on Line 1 of Algorithm 1. The feature vectors are then fed into the function shown on Line 12, which produces a model representing a centroid feature vector. For 1-class learning, we just use the malicious domain model and threshold value  $\Delta$  as shown on Line 8 in Algorithm 2 to determine the label of a candidate domain (*i.e.* feature vector). The 2-class learning method is similar, but the threshold value  $\Delta$  is substituted for a benign domain model.

---

**Algorithm 1:** DRIFT Algorithm 1

---

**Input** :  $O = [o_k^j]$  ( $k$  observations for domain  $j$ ),  $a$  (window size)

**Output:**  $F$  (vector of feature values  $f_i^j$ )

```

1 Function buildFeatureVector ( $O, a$ )
2    $sum = 0$ 
3   for  $i = 1$  to  $k$  do
4      $f_i^j = |f_{i+a}^j - f_{i-a}^j|/2a$ 
5      $sum = sum + f_i^j$ 
6   end
7   for  $i = 1$  to  $k$  do
8      $f_i^j = f_i^j / sum$ 
9   end
10  return  $F$ 

```

**Input** :  $F$  (vector of  $t$  feature values  $f_i^j$ )

**Output:**  $\mathcal{M}$  (vector of  $k$  centroids  $m_i$ )

```

12 Function buildModel ( $F$ )
13    $sum_m = 0$ 
14   for  $i = 1$  to  $k$  do
15      $sum_f = 0$ 
16     for  $j = 1$  to  $t$  do
17        $sum_f = sum_f + f_i^j$ 
18     end
19      $m_i = (1/t) \times sum_f$ 
20      $sum_m = sum_m + m_i$ 
21   end
22   for  $i = 1$  to  $k$  do
23      $m_i = m_i / sum_m$ 
24   end
25   return  $\mathcal{M}$ 

```

---

---

**Algorithm 2:** DRIFT Algorithm 2

---

**Input** :  $\mathcal{M}$  (model),  $f^x$  (feature for candidate domain  $x$ )**Output:**  $D$  (distance)1 **Function** *distance* ( $\mathcal{M}, f^x$ )2      $sum = 0$ 3     **for**  $i = 1$  **to**  $k$  **do**4          $sum = sum + |m_i - f_i^x|$ 5     **end**6     **return**  $sum$ 

7

**Input** :  $\mathcal{M}$  (model),  $f^x$  (feature candidate),  $d$  (threshold)**Output:**  $l$  (label)8 **Function** *OneClassLearning* ( $\mathcal{M}, f^x, d$ )9     **if**  $distance(\mathcal{M}, f^x) > d$  **then**10         **return** “malicious”11     **else**12         **return** “benign”13     **end**

14

**Input** :  $\mathcal{M}, \mathcal{B}$  (benign model),  $f^x$  (feature candidate)**Output:**  $l$  (label)15 **Function** *TwoClassLearning* ( $\mathcal{M}, \mathcal{B}, f^x$ )16     **if**  $distance(\mathcal{B}, f^x) > distance(\mathcal{M}, f^x)$  **then**17         **return** “malicious”18     **else**19         **return** “benign”20     **end**

---

In terms of computational complexity, the building of feature vectors is  $O(k)$  where  $k$  is the number of observations for a given domain  $j$ . Building a model of centroid feature vectors, on the other hand, requires  $O(kt)$  where  $t$  is the number of domains that the model represents. Note that building models of centroid feature vectors for the malicious and benign domains is only a pre-processing step prior to the actual learning phase which runs in  $O(k)$  time. Thus, the overall computational complexity of DRIFT is very low since the main learning phase is linear time.

## 4.6 Threat Model and System Overview

In this section, we describe the objective and the approach taken by adversaries that DRIFT would like to address as well as an overview of the system.

### 4.6.1 Threat Model

With botnets quickly becoming the one of the most prevalent threats on the Internet, the key threat we are most concerned with is ultimately the botmaster who commands a herd of bots. To communicate with their bots, botmasters typically use domain names as its C2 channel because they are easy to acquire and recycle [69]. Since using a single domain name for C2 communication can constitute a single point of failure (*i.e.* law enforcement take-downs), botnet authors (*e.g.* Conficker, Torpiq, Kraken etc.) began adopting algorithmically-generated domain names. These algorithms that generate domain names typically use a pseudorandom number generator (PRNG) that is seeded with a time value. As this algorithm is shared among bots participating in the botnet, using a synchronized seed value such as the current time will allow each bot to generate a similar list of DGA domain names to query in a sequential fashion. In the meantime, a botmaster knowing full knowledge of the potential DGA domain names for any given time can easily register one of those domain names in the DNS. Given that these DGAs can produce hundreds of domain names a day (250 in the case of Conficker A & B), it is becomingly difficult to deduce and reverse-engineer these algorithms due to high entropy of its output. To this end, DRIFT aims to sever the C2 communication link between a botmaster and its bot herd by proactively identifying DGA domain names before they are registered.



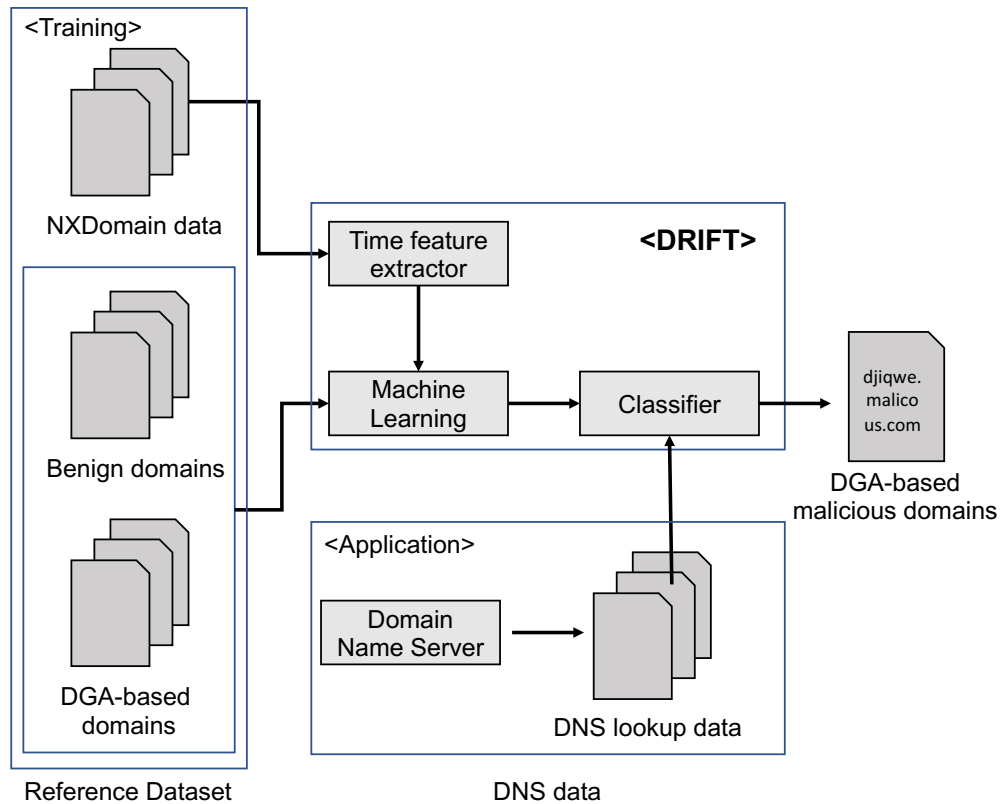


Figure 4.5: A system diagram for the detection of DGA-based malicious domains.

#### 4.6.2 System Overview

The structure of DRIFT is shown in Fig. 4.5, which is a malicious domain detector based on the supervised learning technique discussed in Section 4.5.3. The system is comprised of two stages, a training stage and an application stage. In the training stage, feature extraction and learning are performed using pre-collected data. Afterwards, the application stage classifies DGA-based domain names through the learned classifier.

**Training stage.** The data elaborated on earlier in section 4.4 is used for training as ground-truth data. As mentioned previously, a NXDomain response is returned if there is a connection attempt

from a bot to a DGA domain that is not yet registered in the DNS. The time features of the collected NXDomain responses are extracted and used for supervised learning along with the labels of the domain name (benign or malicious). By analyzing the change of the NXDomain volume over time, the DRIFT system learns about the unique pattern evident of DGA-based botnets.

**Application stage.** Through the training process, the system has a general understanding of the volume of changes in NXDomain responses caused by the clock drift between systems as shown in Figure 4.3. Based on the learned result, it becomes possible to proactively distinguish the queries for the DGA domain names among all queries coming into the DNS in real time. In other words, at this point, the DRIFT system connected to the DNS is able to detect the suspicious domain names which could potentially be used for C2 channel communication in botnets.

## 4.7 Evaluation

To evaluate the performance of our scheme, we use the dataset described in Section 4.4, with the head of the distribution of the dataset corresponding to malicious domains, and the rest of the distribution corresponding to benign domains. With labels known in advance, we proceed to evaluate the labeling capability of our scheme. For 1-class learning, and based on the distribution of the various malicious domains, we set  $\Delta = 0.08$ , which corresponds to 99% of detection accuracy of all the domain name samples considered and included for building the baseline model  $\mathcal{M}$ . To build the model  $\mathcal{M}$ , and to simulate a real-world scenario, we use 1,000 domains. For the unit  $a$ , we calculate the number of queries every hour, and consider a sliding window size  $W$  as 8, 16, 24, 36, and 48 hours (thus, a window of size 24 would move a step of 1 hour at a time to simulate lazy learning of a new difference vector). We start “observing” responses for each 5 days (as highlighted in our dataset) before the registration of a domain. For our evaluation, we consider a variety of evaluation metrics:

### Standard metrics.

- (i) True positives ( $T_p$ ): domains correctly identified as malicious.
- (ii) False positives ( $F_p$ ): domains *incorrectly* marked as malicious.
- (iii) True negative: ( $T_n$ ) domains marked correctly as not malicious.
- (iv) False negative ( $F_n$ ): domains *incorrectly* marked as not malicious.

Using those outcomes, precision, recall, accuracy, and F1 score are  $P = \frac{T_p}{T_p+F_p}$ ,  $R = \frac{T_p}{T_p+F_n}$ ,  $A = \frac{T_p+T_n}{T_p+T_n+F_p+F_n}$ , and  $F1 = 2 \times \frac{P \times R}{P+R}$ .

**Time.** We use how much in advance (before registration) a domain can be detected as a measure of “proactiveness”.

The results for 2-class learning are shown in Table 4.3 and Figure 4.6 across multiple evaluation metrics. We notice that the performance of our scheme is quite promising, especially with a limited amount of knowledge (expressed in a small window size). As for **time** as an evaluation metric, we notice that our scheme can learn with an accuracy of more than 0.90 (on average) for more than 88 hours in advance ( $= 24 \times 5 - 8 - 24$ ) and can achieve an accuracy of more than 0.99 (on average) for more than 48 hours in advance ( $= 24 \times 5 - 48 - 24$ ).

Table 4.3: Standard measurements of performance: true positive, true negative, false positive, false negative for different windows size (average, over 24 slides for the given  $W$  size).

$W$	$T_P$	$T_N$	$F_P$	$F_N$	$P$	$R$	$A$	$F1$
8	91.3	89.4	10.6	8.7	0.89	0.91	0.90	0.90
16	97.4	92.7	7.3	2.6	0.93	0.97	0.95	0.95
24	98.1	94.5	5.5	1.9	0.95	0.98	0.96	0.96
36	99.3	95.5	4.5	0.7	0.96	0.99	0.97	0.98
48	99.4	98.3	1.7	0.6	0.98	0.99	0.99	0.99

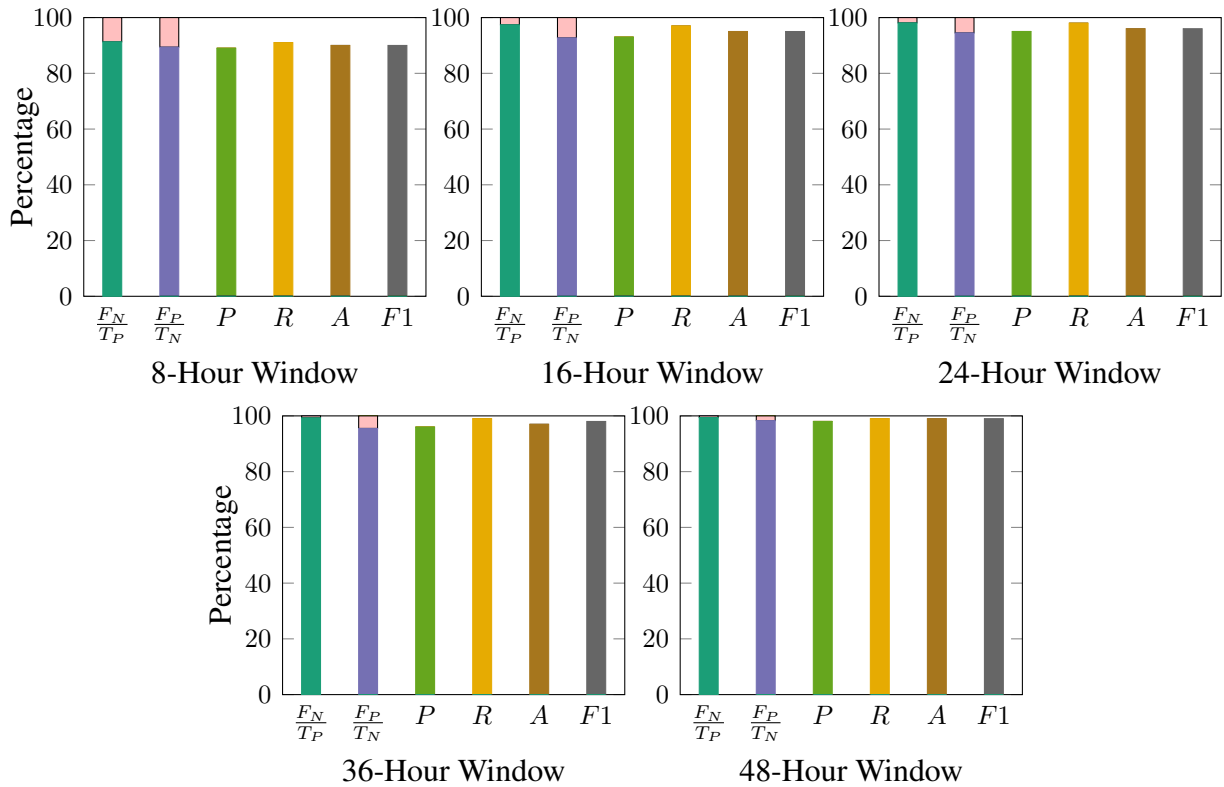


Figure 4.6: Plotting the standard measurements of performance: false negative/true positive, false positive/true negative for different windows size (average, over 24 slides for the given  $W$  size). Notice that an accuracy of 0.99 can be achieved for more than 48 hours in advance.

#### 4.8 Discussion

In this section we cover possible tactics that bots and botmasters can employ to circumvent the DRIFT system. As evident in the Conficker NXDomain DNS dataset which clearly shows significant traffic volume before and after a Conficker-based DGA domain is registered (see Figure 4.3), we attribute this distinguishing pattern to a phenomena known as “clock drift” as outlined in section 4.3.2. Since DGAs tend to use time as a seed into a pseudorandom number generator (PRNG) to ultimately generate a list of domains to query for C2 communication, any deviation from the actual time will cause the infected hosts to query domains prematurely (or late).

#### 4.8.1 Synchronizing to Internet Time

Since DRIFT relies on the fact that DGA-domains will be queried pre and post their registration date, one simple tactic botnet authors can employ to evade DRIFT is to ensure that the time value used for the PRNG seed originates from a *reliable* remote host--such as an internet time server. For example, the botnet author can include code that initiates a HTTP request to a NIST (The National Institute of Standards and Technology) time server located at “<http://nist.time.gov/actualtime.cgi>” which returns the number of milliseconds since Jan 1, 1970.

Indeed the above approach is the case for the Conficker family of malware to obtain the time. As noted by [95], “(Conficker) randomly selects one of six search engines (w3.org, ask.com, msn.com, yahoo.com, google.com and baidu.com)”. By initiating a HTTP GET request to one of these websites, the Conficker DGA can extract the date string GMT from the HTTP header. No matter what time of day, multiple HTTP requests to these websites will have the same result since the DGA only uses the values from the day, month, and year [95]. As Leder *et al.* [75] point out, the selection of these highly-ranked websites to obtain the current time makes it infeasible to launch a coordinated effort to disable all of them at the same time.

If Conficker queries one of the 6 high-profile websites mentioned previously for the actual time, *how is it possible that there were NXDomain queries prior to the registration of a Conficker DGA domain name on a given day?* This may indicate that an incorrect time was returned from one of those high-profile websites, which is highly unlikely. One plausible scenario could be an incorrect or Network Time Protocol (NTP) configuration, especially with virtual machines that typically run web servers. As evidenced on the official NTP.org known issues [8]: “NTP server was not designed to run inside of a virtual machine. It requires a high resolution system clock, with response times to clock interrupts that are serviced with a high level of accuracy.” Another unlikely scenario is an attack on the NTP itself, which is highlighted in the work by Malhotra *et al.*[25] where adversaries

can abuse un-authenticated NTP traffic to modify the system clocks on client machines.

The most likely scenario to explain the clock drift in Conficker, despite obtaining a valid time from a reliable internet source, is revealed in the work of Leder and Werner [75] who illustrated an issue in the PRNG based on a re-implementation in the C programming language. Essentially, they discovered that a cross-compiled version (using MinGW) of the DGA “drifts out of sync after a few hundred operations because the *log()* function used by MinGW differs slightly from the implementation in the *msvcrt.dll* used by Conficker”. Because of rounding errors, they noticed that the digit at position  $10^{13}$  was slightly different which ultimately affected pseudo-randomness of the algorithm and thus rendered the DGA unsynchronized.

#### 4.8.2 Circumventing DNS

As stated previously, one of the main mechanisms of DRIFT to proactively identify DGA domain names is the use of the DNS protocol for C2 communication. As such, botnets can simply circumvent DNS (*i.e.* use IP addresses directly) entirely or use another communication protocol (*e.g.* P2P) that does not rely on domain names or DNS resolution to ultimately evade detection by the DRIFT system. In the following, we discuss these circumvention techniques and alternative C2 communication protocols and their implications.

**Hardcoding C2 Addresses.** In the early days of botnets, bot authors would attempt a stealthy approach of avoiding the DNS entirely by embedding the IP address of the C2 server directly into the bot’s source code [65]. As Khattak *et al.* point out, this was a rather naïve approach since we can reverse engineer the bot to expose the C2 server’s address, thus allowing an eventual takedown by law enforcement. Additionally, these C2 server IP addresses can simply be added to access control lists (ACL) by network administrators, which eliminates C2 communication among all the

bots.

Along the same lines, bot authors can also embed the domain names for C2 servers into the bot's source code, which offers more flexibility than embedding the IP address. This is due to the fact that if the C2 server's IP address is taken down, the domain name can be associated with another IP address so the botmaster can resume their malicious campaign without updating the bots themselves. As mentioned earlier, using a single domain name for C2 communication can constitute a single point of failure--which gave rise to the adoption of DGAs. Since the nature of takedown procedures has grown complex over time [4], botmasters are at an advantage when the takedown procedure begins. Once a DGA-based domain is finally taken down, the C2 server has most likely moved to another domain (in a process known as bot-herding) [65].

**DNS NXDomain Hijacking.** The NXDomain response from botnets querying DGA domains prematurely is the key mechanism behind the DRIFT system. With that said, altering or "hijacking" this response would throw off the measurements utilized by DRIFT since it analyzes the number of NXDomain responses over a fixed period of time (*e.g.* hour). As a case in point, a previous study [43] has shown that the DNS servers operated by certain ISPs: "may hijack such responses in an effort to 'assist' users by sending them to a 'search help' page filled with advertisements"--rather than returning the NXDomain response. Disconcerting as it sounds, a recent large-scaled study [40] concluded that NXDomain hijacking by ISPs is quite rare since it only affected 4.8% of the 1.2M network nodes they measured (which were spread across 14k autonomous systems (ASes) in 172 countries).

Another possible way that botmasters can mitigate NXDomain responses (thereby avoiding detection by DRIFT) is to employ caching DNS servers or "rogue" DNS servers on compromised systems. A typical caching DNS server does not contain any domain resource records, it simply resolves DNS queries from clients and caches the answer to respond quickly for future queries. As

Heron [59] describes, adversaries would only need a compromised machine or a server hosted in a country with lax cyber crime laws. Essentially, when a botnet-originated DNS query is accepted by one of these servers, it could avoid returning a NXDomain response and possibly provide a legitimate IP address under its control.

### 4.8.3 Botnet Topologies

**Centralized botnet.** A centralized botnet is a topology with a clear distinction between the botmaster and its participating bots, shown prominently in Figure 4.1. In this topology, the botmaster prepares the C2 channel to issue commands so that bots accessing the channel can receive instructions to take future actions. In addition to HTTP, these commands can be sent through various protocols such as UDP, TCP, IRC, and so on. In the case of using pre-assigned IP addresses for TCP or UDP, the approach by DRIFT in utilizing the characteristics of the DNS NXDomain responses cannot be directly applied here. However, if a centralized botnet does not use HTTP or DNS resolution, the botmaster can be easily be thwarted simply by using the countermeasures we discussed previously to determine the C2 server's IP address. We highlight that DRIFT's approach focuses on DGA-based botnets, which have become increasingly difficult to take down due to the coordination of law enforcement and different agencies [4].

**Decentralized botnet.** In contrast to a centralized topology, decentralized botnets have no obvious commander which makes the C2 communication not concentrated around a specific node (*i.e.* botmaster). Applying DRIFT's detection mechanisms here will likely be difficult, especially if the C2 communication protocol does not use DNS resolution. For example, a typical decentralized botnet uses peer-to-peer (P2P) technology and works by allowing participating bots to relay the commands to each other (rather than from a centralized node). The decentralized topology is free from the single-point-of-failure issues discussed previously, because different parts of the botnet



also functions as C2 servers. In summary, DRIFT will likely perform well for decentralized botnets that use DNS resolution because it takes advantage of patterns in NXDomain responses due to clock drift. However, since the role of the C2 server is distributed, the amount of NXDomain responses that occur will be significantly lower than that of the centralized botnet. In future work, we will use the same approach for the analysis of decentralized botnets using DNS resolution.

#### 4.9 Conclusion

In this chapter, we proposed a system called DRIFT to proactively detect algorithmically-generated malicious domain names typically employed by botnets. We highlighted the fact that DGA domains tend to have a large number of DNS queries prior to registration, resulting in NXDomain responses which is then followed by a gradual overall decline. We then devised a detection algorithm using the notion of the difference function over the number of NXDomain responses for a given domain with a sliding time window. Using DNS traffic gathered from certain TLDs for the pre-calculated list of generated domains by the Conficker malware variants, our detection algorithm was able to achieve 99% accuracy (on average) as early as 48 hours prior to registration.

## CHAPTER 5: DNS FILTERING & EXTRACTION NETWORK SYSTEM

In this chapter, we propose a system called D-FENS (DNS Filtering & Extraction Network System) which will serve as the first line of defense against malicious domain names. Given that the primary objective of D-FENS is to provide real-time predictions of identifying malicious domain names, we sought a machine learning approach that would allow fast and accurate classification. While there are previous studies that have developed classification systems to detect malicious domain names, they often used a traditional machine learning approach which required engineering several features (*e.g.* URL lexical analysis, network traffic, host-based). Some of these features required costly lookups and complex computations which would potentially affect real-time systems [45].

With the recent success in using deep learning to identify domain names created by malware using DGAs (domain generation algorithms) [121, 118], we chose to adopt a similar strategy for successfully detecting other types of malicious domain names such as phishing and typosquatting. Deep learning is essentially an artificial neural network with multiple hidden layers, which has seen more widespread use with the adoption of faster hardware and graphical processing units (GPUs) that significantly reduce computation time. In their seminal article, LeCun *et al.* [74] states that: “the key aspect of deep learning is that these layers of features are not designed by human engineers: they are learned from data using a general-purpose learning procedure”.

In this work, we make the following contributions:

1. We develop and evaluate D-FENS, a system for classifying malicious domain names in real-time.
2. We describe the implementation of D-FENS as a network middlebox that accepts DNS query requests and routes clients to a safe destination if queries are deemed malicious.

## 5.1 Background & Related Work

Previous studies in malicious domain name identification have typically relied on traditional machine learning methods that require the human effort of feature engineering. With the advent of cloud computing and faster hardware with GPUs to accelerate computation, the most recent work that detects malicious domain names has adopted the use of deep learning for classification. In the following sections, we outline the related work that uses feature-based classifiers and new approaches that use deep learning to identify malicious domains.

### 5.1.1 Feature-based Classification

The 2009 work of Ma *et al.* [81] presented a system that classifies URLs solely based on lexical and host-based features and avoids features which requires examining page content by visiting a potentially unsafe URL. For the lexical features, they considered the “bag-of-words” model along with additional attributes like the lengths of both the host-name and URL. For the host-based features, they examined: IP address properties (*e.g.* blacklist status, autonomous system information, etc.), WHOIS properties (*e.g.* registration date, expiration, etc.), domain name properties (*e.g.* TTL value) and geographic properties (*e.g.* country of IP, connection type, etc.). They evaluated four different classifiers (Naive Bayes, Logistic Regression, and Support Vector Machines with linear and radial basis function kernels) over a dataset of 20-30K URLs from various sources and achieved an accuracy of 95-99% when the false positive rate was low.

The 2010 work of He *et al.* [58] is similar to the previous work of Ma *et al.* in that they avoid the large overhead of fetching and examining page content, but rely on extracting features from the URL to build several Markov Chain models. Using a larger dataset of domains from the “.com” top-level domain (TLD), He *et al.* [58] studied: “three classification techniques, logistic regression,

decision tree, and random forest”. Using 7-fold cross-validation, the results of their experiments showed that the random forest classifier performed the best with the Area Under the Receiver Operating Characteristic Curve (AUC) of 0.88819 and a True Positive (TP) rate of 46.463% when the False Positive (FP) rate was 1%. He *et al.* [58] stressed that: “any classifier with FP-rate larger than 1% is not acceptable”.

Also in 2010, Antonakakis *et al.* [28] introduced a dynamic reputation system for DNS which uses historical DNS information which was collected passively from multiple DNS resolvers. To classify domain names, Notos uses clustering algorithms with three groups of features: network-based (*e.g.* total number of IPs, geographical location, autonomous systems, etc.), zone-based (*e.g.* average length of domain, number of TLDs, character frequencies, etc.) and evidence-based (*e.g.* malware associated with the domain name or its IP address). Their results showed the system had high accuracy scores when classifying new malicious domains in the DNS traffic they observed (TP-rate of 96.8% and FP-rate of 0.38%). However, one limitation they point out is that since the system relies on passive DNS data, it cannot classify malicious domain names when they are newly bought (along with new address space) and never used again for malicious activities.

Similar to [28], Bilge *et al.* [34] proposed Exposure in 2011 which also examines passive DNS to detect malicious domain names. Exposure uses a Decision Tree classifier with 15 features divided into four classes: time-based (*e.g.* short-lived domains, similar daily behavior, etc.), DNS answer-based (*e.g.* unique IP addresses and countries, etc.), TTL value-based (*e.g.* average TTL, number of distinct TTL values, etc.) and domain name-based (*e.g.* percentage of numerical characters). Exposure’s classifier was evaluated using a 10-fold cross validation and percentage split (66% for training) method which resulted in a detection rate around 98%. Exposure was also deployed in a real-world ISP network for two weeks where it analyzed and classified 100 million DNS queries, resulting in 3117 unknown malicious domains being detected.

### 5.1.2 Neural-based Classification

To evade detection and takedowns by law enforcement, malware authors have turned to DGAs to produce pseudo-random domain names for use in command and control (C2) communication within botnets [106]. With some malware families, such as the C variant of Conficker which produces up to 50,000 domains per day [96] (a small subset of which is actively probed), DGAs have made it extremely difficult for researchers and law enforcement to sever C2 communications. To combat this, researchers have proposed DGA-based domain name classifiers [29, 94] which uses similar features to the aforementioned domain name classifiers that typically dealt with malicious types of domains like spam and phishing.

Recent work, such as the 2016 study by Woodbridge *et al.* [118], have started adopting the use of deep learning networks to aid in the identification of DGA-based domain names. In their paper, they present a feature-less approach that leverages Long Short-Term Memory networks (LSTMs) to classify DGA-based domain names in real-time. To train their neural-based LSTM classifier, they used a dataset consisting of the Alexa top 1M (for non-DGA domains) and the DGA domain feed from [1] containing 750,000 samples from thirty DGA families. Their experimental results showed that their LSTM classifier identified 90% of DGAs with a FP-rate of 0.01%.

Similar to Woodbridge *et al.*, the 2017 study by Yu *et al.* [121] also uses deep learning techniques to classify DGA-based domain names. Instead of relying on open datasets such as the Alexa top 1M and synthetically-generated DGA domains, the authors used proprietary data from a real time stream provided by Farsight Security [3]. In addition to a LSTM neural network, they also train a Convolutional Neural Network (CNN) and achieve a 40.31% TP-rate when the FP-rate was 0.01%.

Most recently, the early 2018 work by Lison *et al.* [79] presented a deep learning model that classifies both domain names and IP addresses into 3 classes: benign, malicious, or sinkholes.

Their model was trained on a large passive DNS database provided by Mnemonic [7] (not freely available to the public) which contained over 567 million DNS queries over four years. They extracted a total of 171M distinct domain names and 17M IP addresses. Rather than using the domain name as the sole input to the neural network, the authors also used numerical (*e.g.* number of TTL changes, lifespan of the DNS record, etc.) and categorical (*e.g.* ISP associated with the IP address or the TLD) features as inputs. While the domain names were fed into a Recurrent Neural Network (RNN), the categorical features were fed into an embedding layer and ultimately all three inputs were fed into dense feed-forward layers to produce an output. Their evaluation results demonstrate that their model is capable of detecting 95% of the malicious hosts with a false positive rate of 0.1%.

## 5.2 D-FENS System Model

In Figure 5.1, the system architecture of D-FENS depicts a Training Phase and a Live Phase that acts as a custom DNS Server. In the Training Phase, the Domain Names Collector aggregates malicious and benign domain names from several different sources (see 5.3.1) into a database to ensure there are no duplicates. Labeled data is then fed into the Learning Module which performs deep learning on the data for multiple iterations to produce a model that is loaded by the D-FENS Classifier in the Live Phase discussed below.

### 5.2.1 Implementation

Both phases of D-FENS was primarily written using the Python2 scripting language and a suite of open-source libraries. The Live DNS Server employed the use of Python's built-in web server capabilities to handle HTTP requests and DNS queries were handled by dnslib [19] to encode/decode

DNS wire-format packets. The Learning Phase employed the use of Keras [23], a high-level neural networks API that runs on top of the popular machine learning framework TensorFlow [22] that does numerical computation using data flow graphs. The model was trained on an Amazon Elastic Compute Cloud (Amazon EC2) virtual server instance with access to GPUs (graphics processing units) for accelerated deep learning computation.

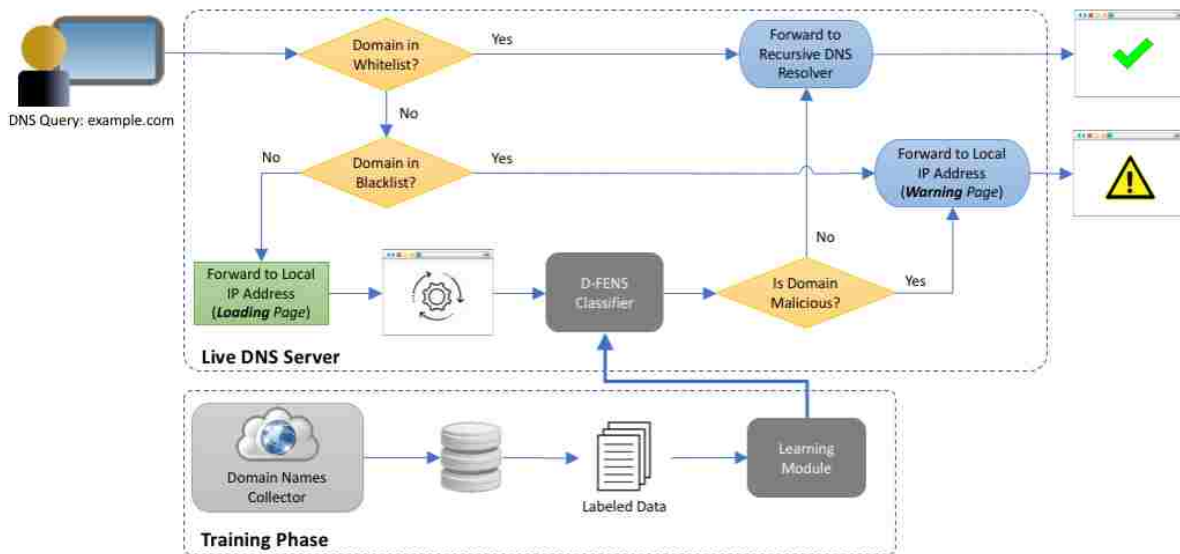


Figure 5.1: An overview of the D-FENS architecture. The Training Phase aggregates domain names and feeds labeled data into the Learning Module, which produces a model for the Classifier in the Live DNS Server Phase. The Live DNS Server accepts DNS queries and checks the current whitelist/blacklist before entering the D-FENS classifier to determine maliciousness.

### 5.2.2 Live DNS Server

D-FENS will ideally be located between the end user and the recursive DNS resolver so it can analyze and process all incoming DNS queries. As shown in Figure 5.1, a DNS query originating from an end user will first be checked against a whitelist of domain names and proceed to the recursive DNS resolver if the DNS query’s domain name has been white-listed. If not, then it proceeds to

check if the queried domain name appears in a blacklist and will forward to a local IP address if it has been blacklisted. The local IP address will be a webserver module controlled by D-FENS and will display a webpage indicating to the end user that the requested domain name is considered malicious. The user will also have an option to add the current domain name to the whitelist so future DNS queries will be sent upstream to the recursive DNS resolver. Following the blacklist check, the domain name is sent to the D-FENS Classifier to determine its maliciousness probability. If D-FENS determines the queried domain name is not malicious, then the DNS request is sent upstream to the recursive resolver as normally to return the appropriate DNS response. Conversely, a domain name that D-FENS classifies as malicious will return a DNS response to redirect the user to a local webserver.

### 5.2.3 *Deep Learning Model*

Our deep learning model employs the use of two mainstream architectures that have been typically used with sentence and document modeling: Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). As described by Zhou *et al.* [126], you can achieve excellent performance and classification results when you combine the strengths of both of these architectures by feeding the output of the CNN into the RNN. Similar to the architecture proposed in [66], we build our deep learning model with an Embedding Layer following by a 1-dimensional CNN which is then fed into LSTM recurrent neural network.

#### 5.2.3.1 *Embedding Layer*

The first step of our deep learning model uses the Embedding Layer, which is a common approach in Natural Language Processing (NLP) for representing words and documents as dense vectors. Unlike traditional approaches which produce sparse vectors composed of mostly zeros (*i.e.* one-hot



encoding), the embedding layer turns the input integers (indexes) into dense vectors of fixed size for computational efficiency. As noted by [36], “the Embedding Layer is initialized with random weights and will learn an embedding for all of the words”, this is specifically for our model while it trains. It is also possible to initialize the weights of the Embedding Layer to incorporate other approaches of word embeddings such as Word2Vec [83] and GloVe [93].

In preparation for the Embedding Layer, we must first convert the raw domain name strings into a sequence of integers corresponding to each character in the domain. Using the index returned from Python’s lookup table of 100 printable characters [9], we systematically convert each character into a unique integer. Since each input must be the same length, we use Keras’ `pad_sequences` preprocessing utility to pad each input with zeros. We chose a length of 87 since the label of the domain name is limited to 63 characters [84] and the longest TLD currently in existence is 24 characters long [6]. For the embedding size, we used a 32-dimensional vector but this can be tuned per model since previous approaches like [118] and [121] used a 128-dimensional vector.

### 5.2.3.2 *Convolutional Layer*

The output from the Embedding Layer is fed into a 1-dimensional (“temporal”) CNN, which are a specialized kind of neural network as noted by [52]: “(CNNs) are networks that use a mathematical operation called convolution instead of a matrix multiplication in at least one of their layers”. Since CNNs were found to be very useful in extracting information from raw signals (*e.g.* images, speech), they have also proven to be state-of-the-art when it treats text as raw signals at the character level [123]. Convolution in 1-dimensional operates on two signals where one is the raw “input” signal and the other is the “filter” (also called a kernel) which slides over the input to produce an output (also called a feature map). In terms of image processing, for example, we can apply convolution in 2-dimensions with an image where the kernel can be a 3x3 box of pixels that

calculates the mean. When this kernel is applied to each pixel in the image, the resulting output effectively blurs the image. To automatically learn these filters, CNNs apply backpropagation when it is trained with labeled data. The CNN layer in our model uses 256 kernels, each having a size of 5 with the output activation function using an Exponential Linear Unit (ELU) [42] along with a max pooling operation with a window of size 4.

### 5.2.3.3 *LSTM Layer*

The next layer is a Long Short-Term Memory network, which is an improved version of the Recurrent Neural Network introduced in 1997 by Hochreiter and Schmidhuber [61] that was motivated by the vanishing gradients problem which causes neural networks to stop training. Recurrent networks essentially work by adding a cycle to a feedforward neural network, which is described by [39] where: “they transmit the information in only one direction, forward, from the input nodes, through the hidden nodes, and to the output nodes”. The cycle added by RNNs is effectively a feedback loop that allows previous decisions (stored in the hidden layers) to affect new decisions on the current input. Over long time steps, however, the gradients calculated during each training iteration tend to either go really small (“vanish”) or increase exponentially (“explode”) causing unstable networks. Instead of a single neural network layer used by RNNs, LSTMs use memory cells containing three gates that essentially control the information flowing through the network allowing it to learn over several time steps. The LSTM layer in our model uses 32 LSTM cells with the default tanh activation function followed by the last fully connected (“dense”) layer that uses a sigmoid activation function.

The final model was compiled using the Adam optimizer [67] which combines the advantages of two popular stochastic optimization methods: AdaGradD [47] and RMSProp [60]. Additionally, we used Dropouts between each of the layers described above, which were introduced by Srivas-

tava *et al.* [108] as a technique to “randomly drop units (along with their connections) from the neural network during training to prevent units from co-adapting too much”, thereby preventing the model from over-fitting.

## 5.3 Evaluation

D-FENS was evaluated using real datasets containing 10’s of thousands of both malicious and benign domain names. In the following, we review the breakdown of the datasets and the evaluation metrics that were used for training the deep-learning model.

### 5.3.1 Data Sets

The list of domain names we used were mainly sourced from open and freely available datasets, such as the Alexa Top 1M websites for our benign data set. We restricted to only using the top 1,000 websites from Alexa, due to the probability of a high number of malicious domains that may be present in the top 1M [110]. We collected most of our benign domain names from DMOZ, which is a manually-managed archive of the entire Internet approved by human editors [16]. Our malicious domain names came a few sources, including phishing data from OpenPhish.org and Phishtank.com (which is verified by human editors) as well as malware domain names from malwaredomains.com and malwaredomainlist.com. Table 5.1 shows the breakdown of each data source where the total amount of domain names used for training was 713,465. For typosquatting data, we leveraged the use of the dataset that was made publicly-available<sup>1</sup> by Agten *et al.* from their work described in [26] which also included a small set of benign domains not featured in our other sets.

---

<sup>1</sup><https://distrinet.cs.kuleuven.be/software/typos15/dataset.php>

Table 5.1: The data sets used in model training.

<b>Data Source</b>	<b># of Domains</b>
Phishtank	15,863
Openphish	501
malwaredomains.com	20,907
malwaredomainlist.com	1,080
Agten <i>et al.</i> (malicious)	12,682
<i>Total Malicious</i>	51,033 (7%)
Alexa	1,000
Agten <i>et al.</i> (benign)	117
DMOZ	675,037
<i>Total Benign</i>	676,154 (93%)
<i>Total Domains:</i>	727,187

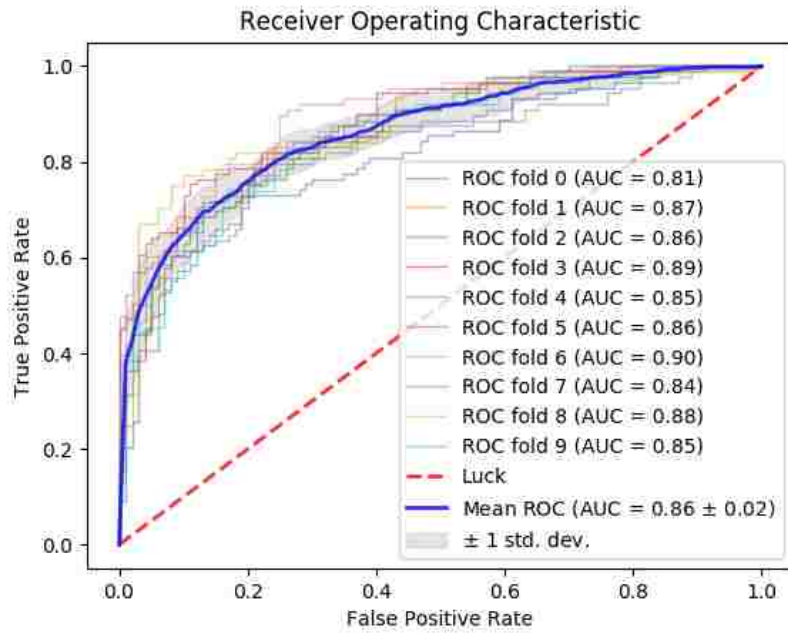
### 5.3.2 Evaluation Metrics

The entire dataset of was evaluated using a 10-fold stratified cross validation method where each model was trained for 10 epochs. The total running time for training the model on AWS EC2 was approximately 18 hours and 20 minutes. We used the Receiver Operating Characteristic (ROC), which is created by plotting the fraction of True Positives vs the fraction of False Positives [24] where the True Positive Rate (TPR) is:

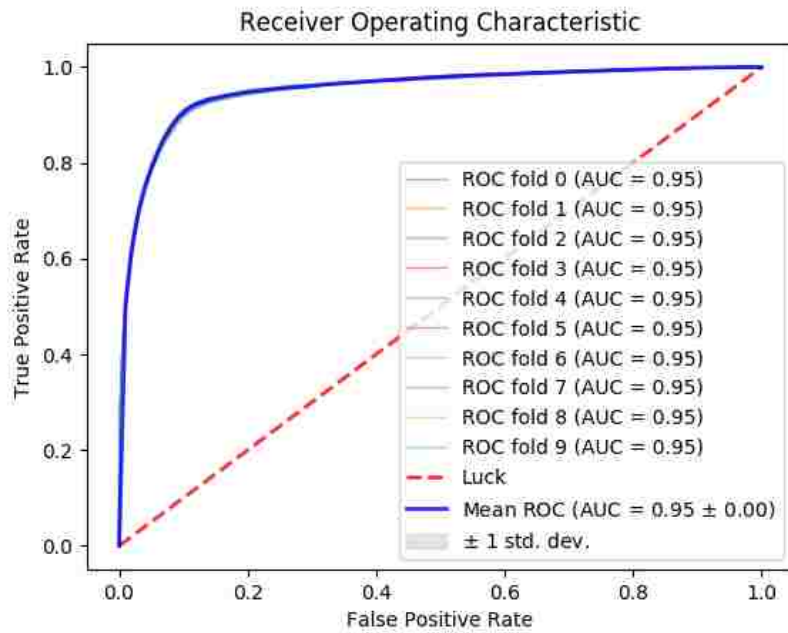
$$TPR = \frac{\sum T_p}{\sum T_p + \sum F_n} \quad (5.1)$$

and the False Positive Rate (FPR) is:

$$FPR = \frac{\sum F_p}{\sum F_p + \sum T_n} \quad (5.2)$$



(a) Small subset of data trained for 1 epoch.



(b) Entire dataset trained for 10 epochs.

Figure 5.2: ROC curves for each fold of the cross validation of the deep learning model.

The area under the ROC curve is known as the AUC, which should be greater than 0.5 for a model to be acceptable (*i.e.* a AUC of 0.5 is no better than chance). Essentially, the AUC area is a measure of predictive accuracy of our model. After running our model for 10 epoch iterations through 10 folds of the cross validation, we achieved a mean AUC value of 0.95 as shown in 5.2(b). For comparison, early results using a small subset of the data with only 1 epoch is shown in 5.2(a).

Once the learning phase completes, the model is loaded by the Classifier and used to make predictions in real-time when answering DNS requests. We tested the live DNS Server by deploying it on a local machine (MacBook Pro 2.3 GHz Intel Core i5 with 8GB) and ran several `nslookup` requests. As shown in Figure 5.3, the elapsed times for several predictions ranged from 2.5 - 3.1 milliseconds.

```
14:46:47: 3.05104255676 milliseconds for prediction
2018-06-08 14:46:47 [DNSHandler:Resolver] Reply: [127.0.0.1:52110] (udp) / 'nytimes.com.' (A) / RRs: A,A,A,A
2018-06-08 14:47:01 [DNSHandler:Resolver] Request: [127.0.0.1:56910] (udp) / 'reddit.com.' (A)
14:47:01: 2.90203094482 milliseconds for prediction
2018-06-08 14:47:01 [DNSHandler:Resolver] Reply: [127.0.0.1:56910] (udp) / 'reddit.com.' (A) / RRs: A,A,A,A
2018-06-08 14:47:18 [DNSHandler:Resolver] Request: [127.0.0.1:57861] (udp) / 'ucf.edu.' (A)
14:47:18: 3.06010246277 milliseconds for prediction
2018-06-08 14:47:18 [DNSHandler:Resolver] Reply: [127.0.0.1:57861] (udp) / 'ucf.edu.' (A) / RRs: A
2018-06-08 14:48:24 [DNSHandler:Resolver] Request: [127.0.0.1:64795] (udp) / 'this-is-a-bad-domain.com.' (A)
14:48:24: 3.00908088684 milliseconds for prediction
14:48:24: >>>> Malicious Domain: this-is-a-bad-domain.com (69.27 % probability)
2018-06-08 14:48:24 [DNSHandler:Resolver] Reply: [127.0.0.1:64795] (udp) / 'this-is-a-bad-domain.com.' (A) / RRs: A
```

Figure 5.3: Sample prediction times for DNS queries showing elapsed times from 2.5 to 3.1 milliseconds.

## 5.4 Summary & Future Work

There were several challenges when attempting to develop a real-time system capable of identifying malicious domain names in a timely fashion. For example, DNS resolution latency delays can range from 1ms (locally cached results) to several seconds [2]. As such, it is crucial that such a system is efficient so far as to not introduce additional delays for the end user while also still striving for accurate classification of malicious domain names. To this end, we proposed a system called

D-FENS which identifies malicious domain names in real-time and operates under two phases: Training and Live prediction that runs inside a DNS server. Rather than identifying features to be used in a traditional machine learning approach, we opted for a deep learning approach which automatically learns the features from the input data. The trade-off is that while the training phase is typically longer than traditional machine learning approaches, execution of live predictions is fast which ranges from 2.5 - 3.1 milliseconds on consumer hardware.

**Future work.** The current results of evaluating the prediction model yielded a mean AUC value of 0.95, after running 10 folds of 10 epoch iterations. We would like to continue running the model for several more epoch iterations and evaluate the results, which could lead to an improved score and more accurate predictions. In addition, we could improve our model by possibly sourcing more malicious domain name data since a large percentage of our existing data was comprised of the DMOZ benign domains. As such, we envision the Training Phase of D-FENS to incorporate a dynamic data collector which constantly updates itself to be aware of the latest reported malicious domains.

## CHAPTER 6: CONCLUSION

Internet users heavily depend on the DNS infrastructure to provide a fast and reliable method to resolve human-readable domain names to IP addresses, which may not always result in benign destinations. Our proposed system called D-FENS will attempt to alleviate this malicious course of action by redirecting users to a safe destination through the use of machine learning and standard lookups in white/black-lists. To our knowledge, this will be the first system that attempts to redirect users away from malicious domain names in real-time by using a hybrid approach of white/black-lists and deep learning.

At the beginning of this research, a comprehensive study on the landscape of the malicious domain name type known as typosquatting was conducted. Several techniques of typosquatting were identified in the literature as well as certain features of domain names that lend to the probability of them being typosquatted. To measure the effectiveness of such techniques and how “successful” they are in deceiving users, we conducted a multi-phase user study that exposed participants to several URLs in which a subset were deliberately modified using known typosquatting techniques. Our results showed that participants generally performed better and faster at identifying typosquatted domain names after being educated about their techniques. We also discovered which typosquatting models were more “successful” than others in deceiving users, and attributed it to studies in Cognitive Science that highlighted how the human brain encodes jumbled characters in words.

Following this trend, we also explored the possibility of an Adversary to take a user’s context into account when devising an attack strategy involving malicious domain names. The results of the user study mentioned previously showed that while some typosquatting techniques fared better than others in deceiving users, the actual selection of the target domain names to typosquat may



play a role. For example, the most incorrectly identified domain name in the study included a ccTLD from Ukraine (.ua) which is not well-known for the average Internet user in the United States. To this end, we designed another user study to establish context by dynamically fetching domain names based on user input (*e.g.* country, subject interest, favorite websites) and similarly applied a random typosquatting model. We found that the most successful typosquatting strategy for an Adversary is to remove inner-character in domain names that users are most familiar with.

Further research brought us into an area of malicious domain names that were not strictly used for deceiving users, rather for a communication channel between a botmaster and its botnet. To avoid detection and takedowns by law enforcement, botnet authors started using domain generation algorithms (DGA) to produce pseudo-random domain names for use in C2 communication. Previous efforts to identify and prevent DGA-domain names from operating have relied on reverse-engineering and intrinsic analysis. We proposed a system called DRIFT to proactively detect DGA-based domains by using DNS query analysis to look for tell-tale patterns of NXDomain responses. Ultimately, we achieved a 99% accuracy on average as early as 48 hours prior to the registration of a malicious DGA-based domain name.

Building upon these previous identification approaches and methods, we ultimately implemented a system to detect malicious domain names in real-time. Our system called D-FENS uses deep learning to produce a model that is used in a live DNS server to accurately predict malicious domain names in DNS queries, thereby preventing users from navigating to unintended destinations.

## **APPENDIX : COPYRIGHT INFORMATION**

## ACM Copyright and Audio/Video Release

**Title of the Work:** Understanding the Effectiveness of Typosquatting Techniques

**Author/Presenter(s):** Jeffrey Spaulding:University of Central Florida;DaeHun Nyang:Inha University;Aziz Mohaisen:University of Central Florida

**Type of material:**Full Paper

**Publication and/or Conference Name:** HotWeb'17: Fifth Workshop on Hot Topics on Web Systems and Technologies Proceedings

### I. Copyright Transfer, Reserved Rights and Permitted Uses

\* Your Copyright Transfer is conditional upon you agreeing to the terms set out below.

Copyright to the Work and to any supplemental files integral to the Work which are submitted with it for review and publication such as an extended proof, a PowerPoint outline, or appendices that may exceed a printed page limit, (including without limitation, the right to publish the Work in whole or in part in any and all forms of media, now or hereafter known) is hereby transferred to the ACM (for Government work, to the extent transferable) effective as of the date of this agreement, on the understanding that the Work has been accepted for publication by ACM.

#### Reserved Rights and Permitted Uses

(a) All rights and permissions the author has not granted to ACM are reserved to the Owner, including all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM, Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "[Major Revision](#)" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, (3) any repository legally mandated by an agency funding the research on which the Work is based, and (4) any non-commercial repository or aggregation that does not duplicate ACM tables of contents, i.e., whose patterns of links do not substantially duplicate an ACM-copyrighted volume or issue. Non-commercial repositories are here understood as repositories owned by non-profit organizations that do not charge a fee for accessing deposited articles and that do not sell advertising or otherwise profit from serving articles.

(iv) Post an "[Author-Izer](#)" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("[Submitted Version](#)" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work.

When preparing your paper for submission using the ACM TeX templates, the rights and permissions information and the bibliographic strip must appear on the lower left hand portion of the first page.

The new [ACM Consolidated TeX template Version 1.3 and above](#) automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

*Please copy and paste the following code snippet into your TeX file between `\begin{document}` and `\maketitle`, either after or before CCS codes.*

```
\copyrightyear{2017}
\acmYear{2017}
\setcopyright{acmcopyright}
\acmConference{HotWeb'17}{October 14, 2017}{San Jose / Silicon Valley, CA,
USA}\acmPrice{15.00}\acmDOI{10.1145/3132465.3132467}
\acmISBN{978-1-4503-5527-8/17/10}
```

ACM TeX template .cls version 2.8, automatically creates and positions these text blocks for you based on the code snippet which is system-generated based on your rights management choice and this particular conference.

*Please copy and paste the following code snippet into your TeX file between `\begin{document}` and `\maketitle`, either after or before CCS codes.*

```
\CopyrightYear{2017}
\setcopyright{acmcopyright}
\conferenceinfo{HotWeb'17,}{October 14, 2017, San Jose / Silicon Valley, CA,
USA}
\isbn{978-1-4503-5527-8/17/10}\acmPrice{$15.00}
\doi{https://doi.org/10.1145/3132465.3132467}
```

*If you are using the ACM Microsoft Word template, or still using an older version of the ACM TeX template, or the current versions of the ACM SIGCHI, SIGGRAPH, or SIGPLAN TeX templates, you must copy and paste the following text block into your document as per the instructions provided with the templates you are using:*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [Permissions@acm.org](mailto:Permissions@acm.org).

*HotWeb'17, October 14, 2017, San Jose / Silicon Valley, CA, USA*

*NOTE: Make sure to include your article's DOI as part of the bibstrip data; DOIs will be registered and become active shortly after publication in the ACM Digital Library*

A. Assent to Assignment. I hereby represent and warrant that I am the sole owner (or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Are any of the co-authors, employees or contractors of a National Government?  Yes  No

---

## II. Permission For Conference Recording and Distribution

\* Your Audio/Video Release is conditional upon you agreeing to the terms set out below.

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release?  Yes  No

## III. Auxiliary Material

Do you have any Auxiliary Materials?  Yes  No

## IV. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

We/I have not used third-party material.

We/I have used third-party materials and have necessary permissions.

## V. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part V and be sure to include a notice of copyright with each such image in the paper.

We/I do not have any artistic images.

We/I have any artistic images.

---

## VI. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

- (a) Owner is the sole owner or authorized agent of Owner(s) of the Work;
- (b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;
- (c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;
- (d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;
- (e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and
- (f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

I agree to the Representations, Warranties and Covenants

## Funding Agents

1. National Research Foundation of Korea award number(s): 2016K1A1A2912757

---

DATE: **08/17/2017** sent to mohaisen@ieee.org at **19:08:56**

# IEEE COPYRIGHT AND CONSENT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IEEE and must accompany any such material in order to be published by the IEEE. Please read the form carefully and keep a copy for your files.

## Proactive Detection of Algorithmically Generated Malicious Domains

Mr. Jeffrey Spaulding, Mr. Jeman Park, Prof. Joongheon Kim and Dr. Aziz Mohaisen

2018 International Conference on Information Networking (ICOIN)

## COPYRIGHT TRANSFER

The undersigned hereby assigns to The Institute of Electrical and Electronics Engineers, Incorporated (the "IEEE") all rights under copyright that may exist in and to: (a) the Work, including any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work; and (b) any associated written or multimedia components or other enhancements accompanying the Work.

## GENERAL TERMS

1. The undersigned represents that he/she has the power and authority to make and execute this form.
2. The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
3. The undersigned agrees that publication with IEEE is subject to the policies and procedures of the [IEEE PSPB Operations Manual](#).
4. In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing copyright transfer shall be null and void. In this case, IEEE will retain a copy of the manuscript for internal administrative/record-keeping purposes.
5. For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.
6. The author hereby warrants that the Work and Presentation (collectively, the "Materials") are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the author has obtained any necessary permissions. Where necessary, the author has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IEEE

**You have indicated that you DO wish to have video/audio recordings made of your conference presentation under terms and conditions set forth in "Consent and Release."**

## CONSENT AND RELEASE

1. In the event the author makes a presentation based upon the Work at a conference hosted or sponsored in whole or in part by the IEEE, the author, in consideration for his/her participation in the conference, hereby grants the IEEE the unlimited, worldwide, irrevocable permission to use, distribute, publish, license, exhibit, record, digitize, broadcast, reproduce and archive, in any format or medium, whether now known or hereafter developed: (a) his/her presentation and comments at the conference; (b) any written materials or multimedia files used in connection with his/her presentation; and (c) any recorded interviews of him/her (collectively, the "Presentation"). The permission granted includes the transcription and reproduction of the Presentation for inclusion in products sold or distributed by IEEE and live or recorded broadcast of the Presentation during or after the conference.
2. In connection with the permission granted in Section 1, the author hereby grants IEEE the unlimited, worldwide, irrevocable right to use his/her name, picture, likeness, voice and biographical information as part of the advertisement, distribution and sale of products incorporating the Work or Presentation, and releases IEEE from any claim based on right of privacy or publicity.

BY TYPING IN YOUR FULL NAME BELOW AND CLICKING THE SUBMIT BUTTON, YOU CERTIFY THAT SUCH ACTION CONSTITUTES YOUR ELECTRONIC SIGNATURE TO THIS FORM IN ACCORDANCE WITH UNITED STATES LAW, WHICH AUTHORIZES ELECTRONIC SIGNATURE BY AUTHENTICATED REQUEST FROM A USER OVER THE INTERNET AS A VALID SUBSTITUTE FOR A WRITTEN SIGNATURE.

Aziz Mohaisen

13-11-2017

Signature

Date (dd-mm-yyyy)

## Information for Authors

### AUTHOR RESPONSIBILITIES

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at [http://www.ieee.org/publications\\_standards/publications/rights/authorrightsresponsibilities.html](http://www.ieee.org/publications_standards/publications/rights/authorrightsresponsibilities.html) Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

### RETAINED RIGHTS/TERMS AND CONDITIONS

- Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.
- Authors/employers may reproduce or authorize others to reproduce the Work, material extracted verbatim from the Work, or derivative works for the author's personal use or for company use, provided that the source and the IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.
- Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use. The IEEE Intellectual Property Rights office must handle all such third-party requests.
- Authors whose work was performed under a grant from a government funding agency are free to fulfill any deposit mandates from that funding agency.

### AUTHOR ONLINE USE

- **Personal Servers.** Authors and/or their employers shall have the right to post the accepted version of IEEE-copyrighted articles on their own personal servers or the servers of their institutions or employers without permission from IEEE, provided that the posted version includes a prominently displayed IEEE copyright notice and, when published, a full citation to the original IEEE publication, including a link to the article abstract in IEEE Xplore. Authors shall not post the final, published versions of their papers.
- **Classroom or Internal Training Use.** An author is expressly permitted to post any portion of the accepted version of his/her own IEEE-copyrighted articles on the author's personal web site or the servers of the author's institution or company in connection with the author's teaching, training, or work responsibilities, provided that the appropriate copyright, credit, and reuse notices appear prominently with the posted material. Examples of permitted uses are lecture materials, course packs, e-reserves, conference presentations, or in-house training courses.
- **Electronic Preprints.** Before submitting an article to an IEEE publication, authors frequently post their manuscripts to their own web site, their employer's site, or to another server that invites constructive comment from colleagues. Upon submission of an article to IEEE, an author is required to transfer copyright in the article to IEEE, and the author must update any previously posted version of the article with a prominently displayed IEEE copyright notice. Upon publication of an article by the IEEE, the author must replace any previously posted electronic versions of the article with either (1) the full citation to the



IEEE work with a Digital Object Identifier (DOI) or link to the article abstract in IEEE Xplore, or (2) the accepted version only (not the IEEE-published version), including the IEEE copyright notice and full citation, with a link to the final, published article in IEEE Xplore.

**Questions about the submission of the form or manuscript must be sent to the publication's editor.**

**Please direct all questions about IEEE copyright policy to:**

**IEEE Intellectual Property Rights Office, [copyrights@ieee.org](mailto:copyrights@ieee.org), +1-732-562-3966**



## REFERENCES

- [1] —. Bambenek Consulting - Master Feeds. <http://osint.bambenek consulting.com/feeds/>.
- [2] —. DNS Prefetching. <https://www.chromium.org/developers/design-documents/dns-prefetching>.
- [3] —. Farsight Security. <https://www.farsightsecurity.com>.
- [4] —. Guidance for Preparing Domain Name Orders, Seizures & Takedowns. <https://go.icann.org/2H1npLi>.
- [5] —. How to change Internet Time Update interval in Windows 10/8/7. <http://bit.ly/2p3Aufk>.
- [6] —. List of Top-Level Domains. <http://data.iana.org/TLD/tlds-alpha-by-domain.txt>.
- [7] —. Mnemonic - Passive DNS. <https://passivedns.mnemonic.no>.
- [8] —. NTP.org: Known Operating System Issues. <http://support.ntp.org/bin/view/Support/KnownOsIssues>.
- [9] —. Python string - Common string operations. <https://bit.ly/PtEdzf>.
- [10] —. FTC Warns That Rapid Expansion of Internet Domain Name System Could Leave Consumers More Vulnerable. <http://1.usa.gov/1LVjyQ5>, 2011.
- [11] —. The conficker working group. <http://bit.ly/1kAYsJA>, Nov 2012.
- [12] —. History Behind .COM. <http://bit.ly/1Uhdba0>, 2015.
- [13] —. Registrar Accreditation: History of the Shared Registry System. <http://bit.ly/1NWexTL>, 2015.

- [14] —. Root Zone Database. <http://bit.ly/1TBSeck>, 2015.
- [15] —. Wireshark Users Guide. [https://www.wireshark.org/docs/wsug\\_html/](https://www.wireshark.org/docs/wsug_html/), 2015.
- [16] —. About DMOZ. <http://dmoz-odp.org/docs/en/about.html/>, 2016.
- [17] —. How to Avoid Survey Fatigue. <https://www.n-r-c.com/how-to-avoid-survey-fatigue/>, 2016.
- [18] —. Daily DNS Changes and Web Hosting Activity. <http://bit.ly/2i9Lqs0v>, 2017.
- [19] —. dnslib 0.9.7. <https://pypi.org/project/dnslib/>, 2017.
- [20] —. Domain Name Registration Process. <https://go.icann.org/2ymkyN9>, 2017.
- [21] —. TLD Zone File Access Program. <http://bit.ly/1pxjRrv>, 2017.
- [22] —. About TensorFlow. <https://keras.io>, 2018.
- [23] —. Keras Documentation. <https://keras.io>, 2018.
- [24] —. ROC Analysis. [http://mlwiki.org/index.php/ROC\\_Analysis](http://mlwiki.org/index.php/ROC_Analysis), 2018.
- [25] E. B. Aanchal Malhotra, Isaac E. Cohen and S. Goldberg. Attacking the network time protocol. In *Proceedings of The Network and Distributed System Security Symposium (NDSS)*, 2016.
- [26] P. Agten, W. Joosen, F. Piessens, and N. Nikiforakis. Seven Months' Worth of Mistakes: A Longitudinal Study of Typosquatting Abuse. In *NDSS*, 2015.
- [27] M. Andrews. Negative caching of DNS queries (DNS NCACHE). RFC 2308, 1998.

- [28] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. *USENIX Security'10: Proceedings of the 19th USENIX conference on Security*, pages 1–17, 2010.
- [29] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *USENIX Security*, 2012.
- [30] Artsiom Holub and Patrick Colford. The Future is Here - Assaulting the Internet with Mirai. <http://bit.ly/2oSkJrU>.
- [31] A. Banerjee, D. Barman, M. Faloutsos, and L. N. Bhuyan. Cyber-Fraud is One Typo Away. In *IEEE INFOCOM*, 2008.
- [32] T. Barabosch, A. Wichmann, F. Leder, and E. Gerhards-Padilla. Automatic extraction of domain name generation algorithms from current malware. In *Proc. NATO Symposium IST-111 on Information Assurance and Cyber Defense, Koblenz, Germany*, 2012.
- [33] G. T. Becker. Righttime: A real time clock correcting program for ms-dos-based computer systems. *Proceedings of the 24th Annual Precise Time and Time Interval Systems and Applications Meeting*, pages 267–276, 1992.
- [34] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *NDSS*, 2011.
- [35] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel. Exposure: A passive DNS analysis service to detect and report malicious domains. *ACM Trans. Inf. Syst. Secur.*, 16(4):14:1–14:28, 2014.
- [36] J. Brownlee. How to Use Word Embedding Layers for Deep Learning with Keras. <https://bit.ly/2qIHUM7>, 2017.

- [37] W. Chang, A. Mohaisen, A. Wang, and S. Chen. Measuring botnets in the wild: Some new trends. In *ACM ASIACCS*, pages 645–650, 2015.
- [38] B. B. Chaudhuri. A new definition of neighborhood of a point in multi-dimensional space. *Pattern Recogn. Lett.*, 17(1):11–17, Jan. 1996.
- [39] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah. Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks. *CoRR*, abs/1710.02913, 2017.
- [40] T. Chung, D. Choffnes, and A. Mislove. Tunneling for transparency: A large-scale analysis of end-to-end violations in the internet. In *Proceedings of the 2016 Internet Measurement Conference, IMC '16*, pages 199–213, New York, NY, USA, 2016. ACM.
- [41] C. G. Clark. The Truth in Domain Names Act of 2003 and a Preventative Measure to Combat Typosquatting. *Cornell Law Review*, 2004.
- [42] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.
- [43] D. Dagon, C. Lee, W. Lee, and N. Provos. Corrupted dns resolution paths: The rise of a malicious resolution authority. In *Proc. 15th Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2008.
- [44] F. J. Damerau. A technique for computer detection and correction of spelling errors. *CACM*, 1964.
- [45] M. Darling, G. Heileman, G. Gressel, A. Ashok, and P. Poornachandran. A lexical approach for classifying malicious URLs. *Proceedings of the 2015 International Conference on High Performance Computing and Simulation, HPCS 2015*, pages 195–202, 2015.

- [46] N. Davuth and S.-R. Kim. Classification of malicious domain names using support vector machine and bi-gram method. *International Journal of Security and Its Applications*, 7(1):51–58, 2013.
- [47] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
- [48] B. Edelman. Large-Scale Registration of Domains with Typographical Errors. <http://bit.ly/1IEGvql>, 2003.
- [49] B. Edelman. Typosquatting: Unintended Adventures in Browsing. *McAfee Security Journal*, 2008.
- [50] Y. Fu, L. Yu, O. Hambolu, I. Ozcelik, B. Husain, J. Sun, K. Sapra, D. Du, C. T. Beasley, and R. R. Brooks. Stealthy domain generation algorithms. *IEEE Transactions on Information Forensics and Security*, 12(6):1430–1443, 2017.
- [51] D. B. Gilwit. The Latest Cybersquatting Trend: Typosquatters, Their Changing Tactics, and How to Prevent Public Deception and Trademark Infringement. *Wash. UJL & Pol’y*, 2003.
- [52] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [53] J. Grainger and C. Whitney. Does the huamn mnid raed wrods as a wlohe? *Trends in cognitive sciences*, 8(2):58–59, 2004.
- [54] M. Grill, I. Nikolaev, V. Valeros, and M. Rehak. Detecting dga malware using netflow. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015.

- [55] H. Guerid, K. Mittig, and A. Serhrouchni. Privacy-preserving domain-flux botnet detection in a large scale network. In *Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on*. IEEE, 2013.
- [56] F. Haddadi and A. N. Zincir-Heywood. Analyzing string format-based classifiers for botnet detection: Gp and svm. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 2626–2633. IEEE, 2013.
- [57] E.-H. Han and G. Karypis. Centroid-based document classification: Analysis and experimental results. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, PKDD '00*, pages 424–431, London, UK, UK, 2000. Springer-Verlag.
- [58] Y. He, Z. Zhong, S. Krasser, and Y. Tang. Mining DNS for malicious domain registrations. *Proceedings of the 6th International ICST Conference on Collaborative Computing: Networking, Applications, Worksharing*, 2010.
- [59] S. Heron. Working the botnet: how dynamic dns is revitalising the zombie army. *Network Security*, 2007(1):9 – 11, 2007.
- [60] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.
- [61] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [62] J. Jackson. Why computers still struggle to tell the time. <http://bit.ly/2BOW5R2>, 2015.
- [63] S. Keats. What’s In A Name: The State of Typo-Squatting 2007. <http://bit.ly/1mqonpI>, 2007.

- [64] M. T. Khan, X. Huo, Z. Li, and C. Kanich. Every Second Counts: Quantifying the Negative Externalities of Cybercrime via Typosquatting. *IEEE Security and Privacy*, 2015.
- [65] S. Khattak, N. R. Ramay, K. R. Khan, A. A. Syed, and S. A. Khayam. A taxonomy of botnet behavior, detection, and defense. *IEEE Communications Surveys Tutorials*, 16(2):898–924, Second 2014.
- [66] Kilby, Melissa. Featureless Deep Learning for Detection of Malicious URLs. <https://bit.ly/2NjLwc6>, 2017.
- [67] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [68] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. In *2005 IEEE Symposium on Security and Privacy (S P'05)*, pages 211–225, May 2005.
- [69] A. Kountouras, P. Kintis, C. Lever, Y. Chen, Y. Nadji, D. Dagon, M. Antonakakis, and R. Joffe. Enabling network security through active DNS datasets. In *RAID*, pages 188–208, 2016.
- [70] B. Krebs. Dot-cm Typosquatting Sites Visited 12M Times So Far in 2018. <https://krebsonsecurity.com/2018/04/dot-cm-typosquatting-sites-visited-12m-times-so-far-in-2018/>, 2018.
- [71] S. Krishnan and F. Monrose. Dns prefetching and its privacy implications: When good things go bad. In *Proceedings of the 3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.
- [72] J. Kwon, J. Lee, H. Lee, and A. Perrig. Psybog: A scalable botnet detection method for large-scale dns traffic. *Computer Networks*, 97:48–73, 2016.



- [73] L. Laboshin, A. Lukashin, and V. Zaborovsky. The big data approach to collecting and analyzing traffic data in large scale networks. *Procedia Computer Science*, 103:536 – 542, 2017. XII International Symposium Intelligent Systems 2016, INTELS 2016, 5-7 October 2016, Moscow, Russia.
- [74] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436, may 2015.
- [75] F. Leder and T. Werner. Know Your Enemy:Containing Conficker. <http://bit.ly/2ESrRQ1>, 2009.
- [76] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.
- [77] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis. A lustrum of malware network communication: Evolution and insights. In *IEEE Symposium on Security and Privacy*, pages 788–804, 2017.
- [78] A. Liska and T. Gallo. *Ransomware: Defending Against Digital Extortion*. O’Reilly Media, Incorporated, 2016.
- [79] P. Lison and V. Mavroeidis. Neural reputation models learned from passive DNS data. *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, 2018-January:3662–3671, 2018.
- [80] M. Lombardi. Nist: Computer time synchronization. <https://tf.nist.gov/service/pdf/computertime.pdf>, 2000.
- [81] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond Blacklists : Learning to Detect Malicious Web Sites from Suspicious URLs. *World Wide Web Internet And Web Information Systems*, pages 1245–1253, 2009.

- [82] J. Maniscalchi. Conficker.A DNS Rendezvous Analysis. <https://bit.ly/2KTIKvK>, 2009.
- [83] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.
- [84] P. Mockapetris. Domain names: implementation and specification (november 1987). RFC 1035, 2004.
- [85] A. Mohaisen and O. Alrawi. Unveiling zeus: automated classification of malware samples. In *WWW (Comp. Volume)*, pages 829–832, 2013.
- [86] A. Mohaisen, O. Alrawi, and M. Mohaisen. AMAL: high-fidelity, behavior-based automated malware analysis and classification. *Computers & Security*, 52:251–266, 2015.
- [87] T. Moore and B. Edelman. Measuring the perpetrators and funders of typosquatting. In *FC*, 2010.
- [88] M. Mowbray and J. Hagen. Finding domain-generation algorithms by looking at length distribution. In *25th IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Naples, Italy, November 3-6, 2014*, pages 395–400, 2014.
- [89] Y. Nadji, M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee. Beheading hydras: performing effective botnet takedowns. In *ACM CCS*, pages 121–132, 2013.
- [90] P. Mockapetris. Domain names—implementation and specification. RFC 1035, Internet Engineering Task Force, Nov 1987.
- [91] K. B. Paterson, V. A. McGowan, and T. R. Jordan. Effects of adult aging on reading filtered text: Evidence from eye movements. *PeerJ*, 1:e63, 2013.
- [92] Patrick Campbell. Making Wireshark work for high-speed networks. <https://bit.ly/2HeSixe>, 2015.

- [93] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.
- [94] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla. A comprehensive measurement study of domain generating malware. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 263–278, 2016.
- [95] P. Porras, H. Saïdi, and V. Yegneswaran. A foray into conficker’s logic and rendezvous points. In *Proceedings of the 2Nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More, LEET’09*, pages 7–7, Berkeley, CA, USA, 2009. USENIX Association.
- [96] Porras, Phillip and Saidi, Hassen and Yegneswaran, Vinod. Conficker C Analysis. <https://bit.ly/2tOib1e>, 2009.
- [97] C. Roth, M. Dunham, and J. Watson. Cybersquatting; typosquatting Facebook’s \$2.8 million in damages and domain names. <http://bit.ly/1SnahSF>, 2013.
- [98] P. Royal. Analysis of the kraken botnet. *Damballa, Apr, 9, 2008*.
- [99] S. Bortzmeyer. DNS privacy considerations. RFC 7626, 2015.
- [100] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero. Phoenix: Dga-based botnet tracking and intelligence. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 11th International Conference, DIMVA 2014, Egham, UK, July 10-11, 2014. Proceedings*, pages 192–211, 2014.
- [101] R. Sharifnya and M. Abadi. Dfbotkiller: domain-flux botnet detection based on the history of group activities and failures in dns traffic. *Digital Investigation*, 12:15–26, 2015.

- [102] S. Shin and G. Gu. Conficker and beyond: a large-scale empirical study. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 151–160. ACM, 2010.
- [103] S. Shin, G. Gu, N. Reddy, and C. P. Lee. A large-scale empirical study of conficker. *IEEE Transactions on Information Forensics and Security*, 7(2):676–690, 2012.
- [104] R. Sizemore. A tale of two clocks. <https://blogs.msdn.microsoft.com/w32time/2007/10/31/a-tale-of-two-clocks/f>, 2007.
- [105] J. Spaulding, D. Nyang, and A. Mohaisen. Understanding the effectiveness of typosquatting techniques. In *Proceedings of the Fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, HotWeb ’17, pages 9:1–9:8, New York, NY, USA, 2017. ACM.
- [106] J. Spaulding, J. Park, J. Kim, and A. Mohaisen. Proactive detection of algorithmically generated malicious domains. In *2018 International Conference on Information Networking (ICOIN)*, pages 21–24, Jan 2018.
- [107] J. Spaulding, S. Upadhyaya, and A. Mohaisen. The landscape of domain name typosquatting: Techniques and countermeasures. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 284–289, Aug 2016.
- [108] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [109] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *ACM CCS*, pages 635–647, 2009.
- [110] J. Szurdi, B. Kocso, G. Cseh, M. Felegyhazi, and C. Kanich. The Long Tail of Typosquatting Domain Names. In *USENIX Security*, 2014.

- [111] M. Thomas and A. Mohaisen. Kindred domains: detecting and clustering botnet domains using DNS traffic. In *WWW (Companion Volume)*, pages 707–712, 2014.
- [112] A. Wang, A. Mohaisen, W. Chang, and S. Chen. Delving into internet ddos attacks by botnets: Characterization and analysis. In *DSN*, 2015.
- [113] A. Wang, A. Mohaisen, W. Chang, and S. Chen. Revealing ddos attack dynamics behind the scenes. In *DIMVA*, 2015.
- [114] A. Wang, A. Mohaisen, and S. Chen. An adversary-centric behavior modeling of ddos attacks. In *IEEE ICDCS*, pages 1126–1136, 2017.
- [115] T. Wang, X. Hu, J. Jang, S. Ji, M. Stoecklin, and T. Taylor. Botmeter: Charting dga-botnet landscapes in large networks. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*. IEEE, 2016.
- [116] T.-S. Wang, C.-S. Lin, and H.-T. Lin. Dga botnet detection utilizing social network analysis. In *Computer, Consumer and Control (IS3C), 2016 International Symposium on*. IEEE, 2016.
- [117] Y. Wang, D. Beck, and J. Wang. Strider typo-patrol: discovery and analysis of systematic typo-squatting. *USENIX SRUTI*, 2006.
- [118] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant. Predicting Domain Generation Algorithms with Long Short-Term Memory Networks. 2016.
- [119] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan. Detecting algorithmically generated malicious domain names. In *ACM IMC*, pages 48–61, 2010.
- [120] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan. Detecting algorithmically generated domain-flux attacks with dns traffic analysis. *IEEE/Acm Transactions on Networking*, 20(5):1663–1677, 2012.

- [121] B. Yu, D. L. Gray, J. Pan, M. D. Cock, and A. C. A. Nascimento. Inline DGA Detection with Deep Networks. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 683–692, 2017.
- [122] H. Zhang, M. Gharaibeh, S. Thanasoulas, and C. Papadopoulos. Botdigger: Detecting dga bots in a single network. In *Proceedings of the IEEE International Workshop on Traffic Monitoring and Analysis*, 2016.
- [123] X. Zhang, J. J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015.
- [124] Y. Zhang, Y. Zhang, and J. Xiao. Detecting the dga-based malicious domain names. In *International Conference on Trustworthy Computing and Services*. Springer, 2013.
- [125] B. Zhao and P. Liu. Private browsing mode not really that private: Dealing with privacy breach caused by browser extensions. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 184–195, June 2015.
- [126] C. Zhou, C. Sun, Z. Liu, and F. C. M. Lau. A C-LSTM Neural Network for Text Classification. 2015.
- [127] L. Zhu, Z. Hu, J. Heidemann, D. Wessels, A. Mankin, and N. Somaiya. Connection-oriented dns to improve privacy and security. In *2015 IEEE Symposium on Security and Privacy*, pages 171–186, May 2015.
- [128] L. Zhu, Z. Hu, J. Heidemann, D. Wessels, A. Mankin, and N. Somaiya. Connection-oriented dns to improve privacy and security. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 171–186, Washington, DC, USA, 2015. IEEE Computer Society.