

2012

## Detecting Abnormal Behavior in Web Applications

Zi Chu

*College of William & Mary - Arts & Sciences*

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Chu, Zi, "Detecting Abnormal Behavior in Web Applications" (2012). *Dissertations, Theses, and Masters Projects*. Paper 1539623355.

<https://dx.doi.org/doi:10.21220/s2-rycp-n008>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact [scholarworks@wm.edu](mailto:scholarworks@wm.edu).

**Detecting Abnormal Behavior in Web Applications**

**Zi Chu**

**Changzhou, Jiangsu, P.R. China**

**Master of Engineering, Southeast University, 2006**  
**Bachelor of Engineering, Southeast University, 2003**

**A Dissertation presented to the Graduate Faculty  
of the College of William and Mary in Candidacy for the Degree of  
Doctor of Philosophy**

**Department of Computer Science**

**The College of William and Mary**  
**May 2012**

## APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of  
the requirements for the degree of

Doctor of Philosophy



---

Zi Chu

Approved by the Committee, April 2012



---

Committee Chair

Associate Professor Haining Wang, Computer Science  
The College of William & Mary



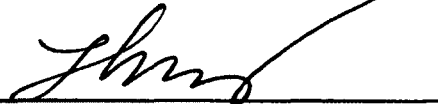
---

Associate Professor Qun Li, Computer Science  
The College of William & Mary



---

Associate Professor Weizhen Mao, Computer Science  
The College of William & Mary



---

Assistant Professor Gang Zhou, Computer Science  
The College of William & Mary



---

Dr. Indra Widjaja  
Bell Laboratories, Alcatel-Lucent

## COMPLIANCE PAGE

Research approved by

Protection of Human Subjects Committee

Protocol number(s): PHSC-2010-05-19-6734-hxwan3

Date(s) of approval: May 31, 2010

## ABSTRACT PAGE

The rapid advance of web technologies has made the Web an essential part of our daily lives. However, network attacks have exploited vulnerabilities of web applications, and caused substantial damages to Internet users. Detecting network attacks is the first and important step in network security. A major branch in this area is anomaly detection. This dissertation concentrates on detecting abnormal behaviors in web applications by employing the following methodology. For a web application, we conduct a set of measurements to reveal the existence of abnormal behaviors in it. We observe the differences between normal and abnormal behaviors. By applying a variety of methods in information extraction, such as heuristics algorithms, machine learning, and information theory, we extract features useful for building a classification system to detect abnormal behaviors.

In particular, we have studied four detection problems in web security. The first is detecting unauthorized hotlinking behavior that plagues hosting servers on the Internet. We analyze a group of common hotlinking attacks and web resources targeted by them. Then we present an anti-hotlinking framework for protecting materials on hosting servers. The second problem is detecting aggressive behavior of automation on Twitter. Our work determines whether a Twitter user is human, bot or cyborg based on the degree of automation. We observe the differences among the three categories in terms of tweeting behavior, tweet content, and account properties. We propose a classification system that uses the combination of features extracted from an unknown user to determine the likelihood of being a human, bot or cyborg. Furthermore, we shift the detection perspective from automation to spam, and introduce the third problem, namely detecting social spam campaigns on Twitter. Evolved from individual spammers, spam campaigns manipulate and coordinate multiple accounts to spread spam on Twitter, and display some collective characteristics. We design an automatic classification system based on machine learning, and apply multiple features to classifying spam campaigns. Complementary to conventional spam detection methods, our work brings efficiency and robustness. Finally, we extend our detection research into the blogosphere to capture blog bots. In this problem, detecting the human presence is an effective defense against the automatic posting ability of blog bots. We introduce behavioral biometrics, mainly mouse and keyboard dynamics, to distinguish between human and bot. By passively monitoring user browsing activities, this detection method does not require any direct user participation, and improves the user experience.

**Dedicated to my family for their love.**

# Contents

<b>Acknowledgments</b>	<b>ix</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xii</b>
<b>1 Introduction</b>	<b>2</b>
<b>2 Hotlinking Investigation and Countermeasures</b>	<b>6</b>
2.1 Problem Statement . . . . .	9
2.1.1 Existing hotlinking techniques . . . . .	10
2.1.2 Defense Against Hotlinking . . . . .	14
2.2 Measurement . . . . .	14
2.2.1 Measurement of Hotlinked Images . . . . .	15
2.2.1.1 Chosen Websites . . . . .	15
2.2.1.2 Data Collection . . . . .	16
2.2.1.3 Data Analysis . . . . .	17

2.2.2	Measurement of Hotlinked Software Packages . . . . .	24
2.2.3	Postmortem Analysis of A Hotlinking Attack . . . . .	26
2.3	Framework Design . . . . .	31
2.3.1	Design Details and Modules . . . . .	31
2.3.1.1	HTTP Request Filtering Module . . . . .	32
2.3.1.2	Session Creation/Authentication Module . . . . .	34
2.3.1.3	Download Authorization Module . . . . .	35
2.3.2	Strict Policy . . . . .	36
2.3.3	Loose Policy . . . . .	37
2.4	Implementation . . . . .	39
2.4.1	Web Server Setup . . . . .	39
2.4.2	Technical Details . . . . .	39
2.5	Evaluation . . . . .	40
2.5.1	Security Analysis . . . . .	40
2.5.1.1	Effectiveness against Direct Hotlinking . . . . .	41
2.5.1.2	Effectiveness against Hotlinking via Referer Fabrication . . . . .	41
2.5.1.3	Effectiveness against Hotlinking via Cookie Vulnerabilities . . . . .	41
2.5.1.4	Effectiveness against Hotlinking via Session Vulnerabilities . . . . .	42
2.5.2	Usability Analysis . . . . .	42
2.6	Related Work . . . . .	44
2.7	Conclusion . . . . .	45
<b>3</b>	<b>Detection of Bots on Twitter</b>	<b>47</b>



3.1	Related Work . . . . .	50
3.2	Measurement . . . . .	53
3.2.1	Data Collection . . . . .	53
3.2.2	Ground Truth Creation . . . . .	54
3.2.3	Data Analysis . . . . .	57
3.3	Classification . . . . .	67
3.3.1	Entropy Component . . . . .	68
3.3.1.1	Entropy Measures . . . . .	68
3.3.2	Spam Detection Component . . . . .	70
3.3.3	Account Properties Component . . . . .	71
3.3.4	Decision Maker . . . . .	73
3.4	Evaluation . . . . .	74
3.4.1	Methodology . . . . .	75
3.4.2	Classification System Training . . . . .	75
3.4.3	Classification System Accuracy . . . . .	76
3.4.4	Twitter Composition . . . . .	78
3.4.5	Resistance to Evasion . . . . .	79
3.5	Conclusion . . . . .	80
<b>4</b>	<b>Detection of Social Spam Campaigns on Twitter</b>	<b>82</b>
4.1	Related Work . . . . .	84
4.1.1	Background of Twitter . . . . .	84
4.1.2	Social Spam Detection . . . . .	86

4.1.3	Scope of This Chapter . . . . .	89
4.2	Characterization . . . . .	90
4.2.1	Data Collection . . . . .	90
4.2.2	Clustering . . . . .	91
4.2.3	Ground Truth Creation . . . . .	92
4.2.4	Campaign Analysis . . . . .	93
4.3	Design . . . . .	98
4.3.1	Classification Features . . . . .	99
4.3.1.1	Tweet-level Features . . . . .	99
4.3.1.2	Account-level Features . . . . .	100
4.3.1.3	Campaign-level Features . . . . .	102
4.3.2	Content Semantic Similarity . . . . .	103
4.3.3	Machine Learning Classifier . . . . .	105
4.4	Evaluation . . . . .	106
4.4.1	Training . . . . .	106
4.4.2	Cross Validation . . . . .	107
4.4.3	System Overhead . . . . .	110
4.5	Conclusion . . . . .	110
<b>5</b>	<b>Detection of Blog Bots via Behavioral Biometrics</b>	<b>112</b>
5.1	Background and Related Work . . . . .	115
5.1.1	Existing Web Bot Detection . . . . .	116
5.1.2	Behavioral Biometrics . . . . .	117

5.2	Behavior Characterization . . . . .	119
5.2.1	Blog Bots . . . . .	119
5.2.2	UI Data Collection . . . . .	122
5.2.3	UI Data Measurements . . . . .	125
5.3	System Design . . . . .	127
5.3.1	Webpage-embedded Logger . . . . .	128
5.3.2	Server-side Detector . . . . .	130
5.3.2.1	Log Processor . . . . .	130
5.3.2.2	Classifier . . . . .	133
5.3.2.3	Decision Maker . . . . .	135
5.4	Evaluation . . . . .	135
5.4.1	Experimental Setup . . . . .	135
5.4.2	System Performance . . . . .	136
5.4.3	System Overhead . . . . .	139
5.5	Discussion . . . . .	140
5.6	Conclusion . . . . .	141
<b>6</b>	<b>Conclusions and Future Work</b>	<b>143</b>
6.1	Conclusions . . . . .	143
6.2	Future Work . . . . .	144
	<b>Bibliography</b>	<b>146</b>
	<b>Vita</b>	<b>156</b>

## ACKNOWLEDGMENTS

This dissertation would not have been accomplished without the support of many people. First and foremost I would like to extend my deepest appreciation to my advisor, Dr. Haining Wang, for his constant guidance with my research and encouragement for my life. It has been a great honor to be his Ph.D. student and friend.

I would like to thank Dr. Qun Li, Dr. Weizhen Mao, Dr. Indra Widjaja and Dr. Gang Zhou for serving on my thesis committee and for their valuable comments and feedback. I would also like to thank the staff of the Computer Science Department and International Student Office for all of their assistance for an international student who came to the States to pursue his studies alone. In particular, I would like to thank Vanessa Godwin, Jacquelyn Johnson and Stephen Sechrist.

My sincere appreciation also goes to fellow graduate students and various friends that have warmly assisted with me in the past five years, Chuan Yue, Steven Gianvecchio, Mengjun Xie, Qi Zhang, Bo Sheng, Ningfang Mi, Yunlian Jiang, Zhenyu Wu, Aaron Koehl, Zhen Ren, Yu He, Jeff Wera, Bo Dong, Feng Yan and others. I feel grateful to have worked with and been inspired by such brilliant people.

Finally, and the most important, I would like to express my deepest gratitude to my parents, Xiaodong Chu and Jijun Yin, to my wife, Yijie Yang, to my aunt and uncle, Heting Chu and Jianmin Shen, and to my global family in Williamsburg, Eugenie and Drayton Hamm, for their love and support in my Ph.D. journey.

# List of Tables

2.1	Category Breakdown by Top-Level Domain . . . . .	16
2.2	Image Hotlinking Distribution per Site Homepage . . . . .	18
2.3	Unique Victim Site Distribution by TLD (16 categories) . . . . .	22
2.4	TLD Distribution of Unique Sites that Hotlinking Software Packages . . . . .	26
2.5	File Storage Log . . . . .	32
2.6	Download Authorization Log . . . . .	35
3.1	Top 10 Tweeting Devices . . . . .	64
3.2	Confusion Matrix . . . . .	77
3.3	Feature Weights . . . . .	79
4.1	A Clustering Example of Semantic Similarity . . . . .	104
4.2	Algorithm Performance Comparison . . . . .	107
4.3	Feature Performance Comparison . . . . .	110
5.1	User Input Actions . . . . .	119
5.2	Classification Features of User Actions . . . . .	132
5.3	True Positive and Negative Rates vs No. of Actions per Group . . . . .	136

**5.4 True Positive and Negative Rates vs Number of Groups . . . . . 138**

# List of Figures

2.1	Basic Forms of Linking Web Objects . . . . .	9
2.2	Fabricating HTTP_REFERERER via PHP . . . . .	10
2.3	HTML snippet exploiting cookie vulnerability . . . . .	12
2.4	HTML snippet of exploiting session vulnerability . . . . .	13
2.5	CDF of the number of hotlinked images per homepage . . . . .	22
2.6	Count of Hotlinked Image without Authorization for top 40 Blog Sites . . . . .	23
2.7	CDF of the number of hotlinked software packages per site . . . . .	25
2.8	Daily traffic in terms of image requests . . . . .	28
2.9	Daily Data Transmission (in MB) Caused by Hotlinking . . . . .	29
2.10	Daily traffic in terms of client IP addresses . . . . .	30
2.11	The hourly traffic distribution of four selected days . . . . .	30
2.12	Anti-Hotlinking Framework Overview . . . . .	32
2.13	Session Creation and Authentication . . . . .	35
2.14	User Download Procedure . . . . .	36
2.15	Modified snippet exploring session vulnerability . . . . .	42
3.1	Numbers of Followers and Friends . . . . .	55

3.2	CDF of Account Reputation . . . . .	57
3.3	Inter-arrival Timing Distribution of Accounts . . . . .	59
3.4	CDF of Tweet Count . . . . .	61
3.5	CDF of Account's Relative Entropy . . . . .	62
3.6	Tweeting Device Makeup . . . . .	63
3.7	External URL ratio in tweets . . . . .	64
3.8	Tweets Posted on Daily/Hourly Base . . . . .	65
3.9	Account Registration Date (Grouped by Quarter) . . . . .	66
3.10	Classification System . . . . .	68
4.1	URL Statistics of Campaigns . . . . .	91
4.2	CDF of Campaign Active Time . . . . .	94
4.3	Inter-arrival Timing Distribution of Campaigns . . . . .	95
4.4	CDF of Entropy of Campaign Posting Inter-arrivals . . . . .	96
4.5	CDF of Account Diversity Ratio of Campaigns . . . . .	98
5.1	Displacement for Point-and-Click . . . . .	120
5.2	Speed for Point-and-Click . . . . .	121
5.3	Movement Efficiency for Point-and-Click . . . . .	125
5.4	Inter-arrival Time Distribution for Keystroke . . . . .	127
5.5	Detection System Architecture . . . . .	129



## **Detecting Abnormal Behavior in Web Applications**

# Chapter 1

The rapid advance of web technologies has made the Web an essential part of our daily lives. People perform a variety of important activities on the web, such as searching, communicating, online shopping, and entertaining. However, network attacks have exploited vulnerabilities of web applications, and caused substantial damages to Internet users. For example, the Internet Security Threat Report by Symantec [133] shows that attacks have increasingly become Web-based. The shift in the motivation of attackers from “attacking for fun” to “attacking for profit” has made attacking techniques more sophisticated. Detecting network attacks is the first and important step in network security. A major branch in this area is anomaly detection.

Anomaly detection refers to detecting patterns that do not conform to the established normal behaviors [74]. Anomaly detection is effective in detecting unknown attacks, which the conventional detection methods often fail to capture. In particular, a typical method of anomaly detection defines a profile representing normal behaviors, and declares any significant deviation from the normal profile as an anomaly. There are some challenges for anomaly detection. First, it is difficult to define a normal profile that can accurately separate normal behaviors from anomalous ones, as the boundary between normal and anomalous behavior is often indistinct. Second, attackers often adapt to make their malicious behaviors close to normal ones.

The focus of this dissertation is on detecting abnormal behaviors in web applications. The basic methodology we used is briefly described as follows. For a web application, we conduct a set of

measurements to reveal the existence of abnormal behaviors in it. We observe the differences between normal and abnormal behaviors. By applying a variety of methods in information extraction, such as heuristics algorithms, machine learning, and information theory, we extract features useful for building a classification system to detect abnormal behaviors.

The major research contributions of this dissertation are summarized below.

(1) Investigation of hotlinking and its countermeasures

Hotlinking is a web behavior that links web resources on a hosting site into a webpage belonging to another site. However, unauthorized hotlinking is unethical, because it not only violates the interests of hosting sites by consuming bandwidth and detracting site visiting traffic, but also violates the copyrights of protected materials. To fully understand the nature of hotlinking, we conduct a large-scale measurement study and observe that hotlinking widely exists over the Internet and is severe in certain categories of websites. Moreover, we perform a detailed postmortem analysis on a real hotlink-victim site. After analyzing a group of commonly used hotlinking attacks and the weakness of current defense methods, we present an anti-hotlinking framework for protecting materials on hosting servers based on existing network security techniques. The framework can be easily deployed at the server side with moderate modifications, and is highly customizable with different granularities of protection. We implement a prototype of the framework and evaluate its effectiveness against hotlinking attacks.

(2) Detection of bots on Twitter

Twitter is a new web application playing dual roles of online social networking and micro-blogging. Users communicate with each other by publishing text-based posts. The popularity and open structure of Twitter have attracted a large number of automated programs, known as bots, which appear to be a double-edged sword to Twitter. Legitimate bots generate a large amount of

benign tweets delivering news and updating feeds, while malicious bots spread spam or malicious contents. More interestingly, in the middle between human and bot, there has emerged cyborg referred to either bot-assisted human or human-assisted bot. To assist human users in identifying who they are interacting with, this chapter focuses on the classification of human, bot and cyborg accounts on Twitter. We first conduct a set of large-scale measurements with a collection of over 500,000 accounts. We observe the difference among human, bot and cyborg in terms of tweeting behavior, tweet content, and account properties. Based on the measurement results, we propose a classification system that includes the following four parts: (1) an entropy-based component, (2) a machine-learning-based component, (3) an account properties component, and (4) a decision maker. It uses the combination of features extracted from an unknown user to determine the likelihood of being a human, bot or cyborg. Our experimental evaluation demonstrates the efficacy of the proposed classification system.

### (3) Detection of social spam campaigns on Twitter

The popularity of Twitter greatly depends on the quality and integrity of the contents contributed by users. Unfortunately, Twitter has attracted spammers to post spam content which pollutes the community. Social spamming is more successful than traditional methods such as email spamming by using social relationship between users. Detecting spam is the first and very critical step in the battle of fighting spam. Conventional detection methods check individual messages or accounts for the existence of spam. Our work takes the collective perspective, and focuses on detecting spam campaigns that manipulate multiple accounts to spread spam on Twitter. As a complement to conventional detection methods, our work brings efficiency and robustness. More specifically, we design an automatic classification system based on machine learning, and apply multiple features to classifying spam campaigns. The experimental evaluation demonstrates the efficacy of the proposed

classification system.

#### (4) Detection of blog bots through behavioral biometrics

Blog bots are automated scripts or programs that post comments to blog sites, often including spam or other malicious links. An effective defense against the automatic form filling and posting from blog bots is to detect and validate the human presence. Conventional detection methods usually require direct participation of human users, such as recognizing a CAPTCHA image, which can be burdensome for users. Our work presents a new detection approach by using behavioral biometrics, primarily mouse and keystroke dynamics, to distinguish between human and bot. Based on passive monitoring, the proposed approach does not require any direct user participation. We collect real user input data from a very active online community and blog site, and use this data to characterize behavioral differences between human and bot. The most useful features for classification provide the basis for a detection system consisting of two main components: a webpage-embedded logger and a server-side classifier. The webpage-embedded logger records mouse movement and keystroke data while a user is filling out a form, and provides this data in batches to a server-side detector, which classifies the poster as human or bot. Our experimental results demonstrate an overall detection accuracy greater than 99%, with negligible overhead.

The remainder of this dissertation is organized as follows. Chapter 2 details our investigation of hotlinking and presents an anti-hotlinking framework for protecting materials on hosting servers. Chapter 3 covers the measurement and classification of bots on Twitter. Chapter 4 describes the detection of social spam campaigns on Twitter. Chapter 5 presents the blog bot detection system based on behavioral biometrics. Finally, Chapter 6 concludes the dissertation and outlines directions for the future work.

## Chapter 2

With the rapid advance of Web technologies, websites not only display text-based information, but also host various types of materials including images, media clips, software installation files, and so on. Those hosted materials of great interest and high value help websites attract users and increase site traffic. To date, more and more websites link web elements provided by third parties. Hotlinking, as a web phenomenon, can be defined as including a linked object (often in the form of an image or document) from one site that actually hosts it into a web page belonging to another site. A hotlinking site has no need to host linked objects itself. In the context of social engineering, hotlinking is a double edged sword. The ethical boundary between benign and malicious hotlinking is whether the linking behavior is authorized by the hosting site or not.

Authorized hotlinking is beneficial to site interaction. For example, a site may include some ad images provided by an advertisement syndicator to make advertising revenue. It does not need to host any ad images by itself, but link them from the syndicator's server. More specifically, the webpage contains a JavaScript snippet provided by the syndicator; during the execution, JavaScript embeds ad links from the syndicator. In this way, the syndicator can dynamically change ad contents and obtain the first-hand trace of ad display. Since the above interaction is approved by both parties, it is a benign hotlinking behavior.

By contrast, unauthorized hotlinking is often harmful to hosting sites. There are some common reasons for unauthorized hotlinking listed as follows. First, some web developers are unprofes-

sional. Their laziness makes them to directly link web objects hosted somewhere else. Second, the hotlinking site may not have enough online storage space to host all the materials it wants to display or accommodate bandwidth demands by frequent visits. Third, the hotlinking site attempts to display some "grey materials" such as pirated media. Hotlinking instead of hosting such materials may dodge legal prosecution.

Considering the simplified website operation model from the economical perspective, the site hosting cost is attributed to paying the hosting service provider for storage space and bandwidth quota, while the gain includes website brand effect and online advertisement revenue. As far as the gain outweighs the cost, this economically sustainable model drives the prosperity of the Internet community. However, the rampant unauthorized hotlinking has recently disturbed the harmonious development of websites, because this unethical behavior violates the interests of hosting sites in terms of the following aspects.

- **Bandwidth theft.** Most websites hosted on third-party hosting servers have to pay for a limited amount of traffic delivery. If the bandwidth consumption exceeds the prepaid quota, the website may be charged more or, in the worst-case scenario, shut down temporarily. It is evident that stealing bandwidth increases the site hosting cost. From this perspective, unauthorized hotlinking is also known as leeching or bandwidth theft [141].
- **Visitor traffic loss.** Many websites hosting free materials rely on online advertising revenue. They display ads assigned by syndicators (like Google AdSense [30] and Yahoo! Advertising [62]) on web pages, and are paid for ad impressions or clicks. If the hosted materials are hotlinked, visitors are directly brought to hotlinking pages. Since legal hosting pages are bypassed, no advertising revenue is generated. Visitor traffic loss could also reduce the site

brand effect. Users view or download popular hotlinked materials through hotlinking sites without knowing the existence of the hosting site.

- **Copyright infringement.** Many web objects (such as photographs and software programs) are copyrighted or licensed. Owners have exclusive rights to display their works publicly [130]. Hotlinking, even in the form of inline linking or framing, such objects in the commercial environment may infringe copyright, and is not justified by fair use [23].

The chapter focuses on the *malicious (i.e., unauthorized)* hotlinking. In the context of a hotlinking attack, we call the site that actually hosts the object *the hosting site* or *victim site*, and the site that hotlinks the object *the hotlinking site* throughout the chapter. To fully understand the nature of hotlinking, we conduct a large-scale measurement study over the Internet, targeting two popular types of web materials, images and software packages. In the image-centric measurement, we choose 1,453 popular websites and check for images hotlinked in their homepages. We find out that about 75.0% of sites hotlink images. To decide whether such hotlinking behaviors are authorized or not, we further analyze the nature of images and their hosting sites. A great amount of images are hosted on special-purposed sites (such as ad syndicators and traffic monitoring sites) and thus should be considered as hotlinked with authorization. However, we also observe that unauthorized hotlinking widely exists in certain categories of websites like blogging. In the software-centric measurements, we select 100 popular software packages as the targets. We search the Internet for websites linking those software binaries, and find out that most hosting sites are hotlinking victims. Since the download link appears in the hotlinking page, the visitor can directly download the software package without visiting the hosting site.

It is a cat-and-mouse game between hotlinking attack and anti-hotlinking defense. The chapter



```
L1: <a href="www.V.com/b.htm">Go to Page b</a>
L2: 
L3: <object>
    <param name="movie" value="V.com/media/d.swf">
    <embed src="V.com/media/d.swf" type="application/x-shockwave-flash"></embed>
</object>
L4: <a href="www.V.com/files/e.pdf"/>Download PDF</a>
```

Figure 2.1: Basic Forms of Linking Web Objects

analyzes a series of commonly used hotlinking attacks and corresponding defense methods. Integrating with the existing anti-hotlinking techniques, we present an anti-hotlinking framework for protecting hosting sites. We also implement a prototype of the proposed framework to demonstrate its feasibility, and evaluate its effectiveness against hotlinking attacks in terms of security and usability. The framework can be easily deployed at the server side with moderate modifications. Moreover, it is highly customizable with different granularities of protection, with which a webmaster can define the different requirements that visitors must complete to qualify for downloading resources.

The remainder of this chapter is organized as follows. We present the threat model along with some common hotlinking attacks in Section 2.1. We describe the large-scale measurement-based study of hotlinking over the Internet in Section 2.2. We detail our anti-hotlinking framework design in Section 2.3. The framework implementation and evaluation are given in Sections 2.4 and 2.5, respectively. Section 2.6 surveys some related work. The chapter concludes in Section 2.7.

## 2.1 Problem Statement

Hotlinking generally involves two parties: hotlinking sites and victim sites (namely original sites or hosting sites). The HTML snippet shown in Figure 2.1 gives an example of hotlinking with four types of regular web objects. Suppose the example page URL of the hotlinking site is [www.H.com/h.htm](http://www.H.com/h.htm).

```
$server = 'www.H.com';  
$host = 'www.V.com';  
$target = '/files/install.rpm';  
$referer = 'www.V.com/'; // Fake $referer injected here  
$fp = fsockopen($server, $port, $errno, $errstr, 30);  
fwrite($fp, $out);
```

Figure 2.2: Fabricating HTTP\_REFERER via PHP

All of the linked objects are hosted by the victim site, V.com. Linking another web page (shown as L1) is not considered as the hotlinking behavior since visitors will be directly brought to that page. L2 to L4 show how to hotlink an image, flash clip and file, respectively.

We assume that the hotlinker owns an independent domain name (such as H.com), and fully operates a web server that hosts his site. We further assume that the hotlinking attack is constrained by the same-origin policy enforced by browsers, since the same-origin policy has been widely used in modern browsers. The policy isolates web contents from different schemes, hosts or ports [99]. For example, the scripts from <http://H.com> cannot access or modify the content of <http://V.com> on the same page.

The objectives of a hotlinker include: (1) hotlinking web objects hosted on other servers into web pages of his own, and (2) ensuring hotlinked objects to be normally displayed or accessed on the client-side. The first goal is easy to achieve by using similar codes shown in Figure 2.1. The second goal is more challenging to reach, since the hotlinker has to detect and bypass the anti-hotlinking mechanisms enforced by hosting sites. In the following, we list some common hotlinking techniques, and analyze how they evade the corresponding defense countermeasures.

### 2.1.1 Existing hotlinking techniques

Directly hotlinking web objects into hotlinkers' web pages (known as *Direct Hotlinking*) is the basic form of hotlinking attacks. It can simply use HTML codes displayed in Figure 2.1. As a

counterattack against *Direct Hotlinking*, many hosting sites currently use a straightforward Referer-based technique. It judges the Referer field in the incoming HTTP request header. If the Referer does not belong to its own domain, the hosting site determines that the requested material is linked from another domain (namely hotlinked) and thus refuses to respond. According to the HTTP protocol specification [35], the Referer field allows the client to identify the URI of the resource from which the Request-URI was obtained. Therefore, checking the Referer field is an effective way to prevent hotlinking as long as the integrity of Referer can be guaranteed.

Hotlinking sites can fabricate the Referer field to bypass the widely used Referer-based defense. Figure 2.2 shows an example of using PHP code to fake HTTP\_REFERER for outgoing HTTP request headers<sup>1</sup>. An arbitrary fake Referer could be injected at the statement `$referer = 'www.Any.com/'`. When a user clicks on the hotlinked download link, the browser generates an HTTP request for that file (`www.V.com/files/install.rpm`), and HTTP\_REFERER is changed into `'V.com/'`, instead of the real one (`H.com/h.htm`). Because the victim server always allows web pages in its own domain to link hosted materials, this type of attacks evades the Referer-based defense. T. Chen et. al [76] and N. Chou et. al [77] confirm in their research of web-based identification theft again that, the Referer field is easy to fabricate.

Many websites have realized the vulnerability of HTTP Referer, and have improved anti-hotlinking methods with the help of cookie and session mechanisms. Cookie is a piece of data that a server stores on the browser to maintain the HTTP communication state between the client and server sides, since the HTTP protocol itself is stateless [35]. If the server receives the cookie issued by itself, it can tell that the user has already visited the site before and thus grants the file download.

Different from cookie, the session mechanism stores information on the server side except a

---

<sup>1</sup>tampering with HTTP request header can also be achieved using other scripts or tools

```
<iframe src="V.com/v.htm" style="display:none">
<a href="V.com/files/install.rpm">Download RMP Package</a>
```

**Figure 2.3:** HTML snippet exploiting cookie vulnerability

session ID on the client side [34]. To implement the session tracking, there are two methods to store session id on the browser: (1) using cookie to record session ID, or (2) URL Rewriting. Since the former method works similarly with the cookie mechanism, we use the latter one as an example to describe how the session mechanism work. When the browser visits `www.V.com/v.htm`, the server creates a session with a unique ID. Suppose that the server implements URL Rewriting via PHP [10]. The session ID will be automatically appended to the links towards the same domain as a URL parameter (such as `www.V.com/v.htm?PHPSESSID=5k32d0`). When the server receives the request, it authenticates the session ID and returns the file if the id is valid.

However, the defense framework purely based on cookie or session mechanism can still be evaded by more sophisticated hotlinking attacks, which are described as follows. In the hotlinking attack against Cookie Protection as shown in Figure 2.3, the hotlinker can include a hidden iFrame on the hotlinking page to bypass the cookie check. When the browser parses the HTML document of `h.htm`, the iFrame will load the page `v.htm` from `V.com`. `V.com` then stores a cookie in the browser. When a user clicks the download link, an HTTP request (`GET /files/install.rpm HTTP/1.1`) is sent to `V.com`, and the cookie issued by `V.com` is also included in the request header. After `V.com` receives its cookie, it returns an HTTP response containing the file `install.rpm` to the browser. Note that the file should be considered as *hotlinked* in `h.htm` because the legal page `v.htm` is hidden in the iFrame and never displayed to the user. [119] and [109] furthermore discuss securing and extending cookie with security properties.

Simple HTML tricks can also be used to bypass the session-based defense (namely Hotlinking

```
<script>function append_sid(){
  retrieves SID;
  var fileLinkObj = document.getElementById("fileLink");
  var newURL = fileLinkObj.getAttribute("href") + "?&PHPSESSID=SID";
  fileLinkObj.setAttribute("href", newURL);
}</script>


<a id="fileLink" href="www.V.com/files/install.rpm" onclick="append_sid();">
  Download RMP Package</a>
```

**Figure 2.4:** HTML snippet of exploiting session vulnerability

against Session Protection). Figure 2.4 shows such an instance. We assume that a session ID is passed to the browser via URL Rewriting. After the browser parses the HTML document of the page h.htm, it sends an HTTP request (GET www.V.com/v.htm HTTP/1.1) attempting to obtain the fictitious image via the URL specified by the href attribute of the <img> tag. When V.com establishes the session with the browser and assigns a session ID, the malicious JavaScript code, append\_sid(), can be triggered to retrieve the session ID and then append it to the href attribute of the file download link with the JavaScript functions getAttribute() and setAttribute(). Thus, the request URL for the file will be artificially rewritten with the legal session ID. Once V.com receives the HTTP request and validates the session ID, it grants the file request and the anti-hotlinking defense becomes void. The legal page v.htm is not displayed because it is included in an <img> tag, and of course fails to render. The naughty script exists in the hotlinking page, and it is beyond the ability of script purifying techniques deployed by benign servers [135]. [117] listed some more complicated session hijacking cases as the authors evaluated the security level of some popular free web-mail.

### **2.1.2 Defense Against Hotlinking**

There are two places to enforce anti-hotlinking, the client side and server side. Since end users do not have direct motivation to defend against hotlinking, the client side is not the ideal place for anti-hotlinking. First, hotlinking is transparent to end users. Very few users would even notice or care whether embedded web materials are from other domains or not. Second, anti-hotlinking may deteriorate user-perceived performance and even interfere with users' browsing activities. For example, some defense methods simply refuse HTTP requests for hotlinked objects, and thus involved web contents fail to load on the browser. By contrast, unauthorized hotlinking steals server resources and costs revenue loss to victim sites. Thus, we believe that the server-side is the appropriate place to deploy the anti-hotlinking framework, and it is the responsibility of hosting sites to protect their materials from being hotlinked. Our proposed anti-hotlinking framework adopts this design principle (see more details in Section 2.3).

## **2.2 Measurement**

In this section we first conduct a large-scale measurement study to understand the nature of hotlinking in a quantitative way. Among the regular types of hotlinked web objects, we choose image and software installation packages (a typical representative of large-sized files) as the measurement targets. Then we observe a hotlinking attack on a web server and perform a detailed analysis based on the logs of the hotlinking victim. Our analysis is focused on the negative effect on the hotlinking victim

### **2.2.1 Measurement of Hotlinked Images**

With the development of Web 2.0, web sites interact with each other much closer than ever before. Web pages of a site may link a variety of materials hosted on third-party sites, such as scripts and ad contents. Among them, images are the most hotlinked. In this part of image-centric measurements, we choose some representative sites, and record images hotlinked in site homepages.

Hotlinking is a simple web technique. Whether it is ethical or not to hotlink depends on how it is used. Hotlinking an image with the authorization from its hosting site is benign and acceptable. For example, the site homepage links an image hosted by a traffic monitor site. Each request for the homepage triggers a request for the hotlinked image. In this way, the monitor site traces visitor traffic for the client site. On the other hand, intentional hotlinking without authorization is unethical. For instance, a blog hotlinks a copyrighted photograph from the official site of a celebrity to attract click traffic. In this case, displaying the image is against the owner's will, and incurs additional transfer traffic for the hosting site.

We cannot determine hotlinking is authorized or not merely based on the URL of the hotlinked image. We analyze the characteristics of images and their hosting sites to classify authorized and unauthorized hotlinking with the help of a set of pre-defined rules. We focus more on unauthorized hotlinking.

#### **2.2.1.1 Chosen Websites**

The target websites for measurement are chosen as follows. We select 15 categories listed by Alexa [5], and take the top 100 sites from each category. These categories are mainly divided based on site content. Alexa is a well-known web archiving site that consistently monitors web traffic and site popularity ranking. We also add a 16th category—blogging. From ranking lists published by

Table 2.1: Category Breakdown by Top-Level Domain

Category	com	net	org	gov	edu	cc	other	Total
Arts	85	3	4	0	0	7	1	100
Business	90	2	0	1	0	7	0	100
Computers	91	1	6	0	0	2	0	100
Games	96	2	2	0	0	0	0	100
Health	61	2	15	14	2	5	1	100
Home	89	1	3	5	0	2	0	100
News	85	0	2	0	0	13	0	100
Recreation	88	2	3	3	0	3	1	100
Reference	37	0	10	6	38	8	1	100
Regional	64	0	2	3	0	31	0	100
Science	49	1	15	19	6	10	0	100
Shopping	95	0	1	0	0	4	0	100
Society	68	1	11	10	1	7	2	100
Sports	86	0	2	0	0	12	0	100
World	38	4	4	0	0	53	1	100
Blogging	319	17	11	9	1	47	5	400
Total	1441	36	91	61	48	211	12	1900
Unique	1115(76.7%)	34(2.3%)	72(5.0%)	28(1.9%)	43(3.0%)	151(10.4%)	10(0.7%)	1453

[18, 20], we include the top 400 blog sites, as many of such sites greatly hotlink images because of their shortages in hosting storage space and bandwidth quota.

Table 2.1 lists the breakdown summary of all the 16 categories by DNS Top-Level Domain (TLD) for comparison. Note that some sites are listed in multiple categories. For example, [www.google.com](http://www.google.com) appears in both the Computers and World categories. Besides, some sites have multiple domain (or sub-domain) names listed (i.e., [news.bbc.co.uk](http://news.bbc.co.uk) and [bbc.co.uk](http://bbc.co.uk)). We run the analysis toolkit to eliminate the duplicates to have the unique site numbers listed at the bottom of Table 2.1. We can see that 1,115 (77%) unique sites belong to the .com TLD, which is the dominant domain in our chosen sites, followed by the country code (denoted as cc) TLD that contributes 151 (10%) unique sites. The rest of TLDs takes up 13% of the chosen sites.

### 2.2.1.2 Data Collection

We develop a Firefox extension along with some script commands to automatically visit the home-pages of target sites one by one. Displaying a webpage on a browser generally involves the fol-



lowing two steps: (1) after the browser loads the page, it parses the HTML document; and then (2) the browser performs a series of actions to display the page on a best-effort basis, including executing dynamic codes (like JavaScript), downloading embedded images, and so on. After the browser loads and parses the page, our Firefox extension logs all the outgoing requests for web objects (like images, JavaScript snippets, .css files, etc) made by the browser in real-time. We believe that this dynamic logging mechanism is more accurate than the traditional static content check over web objects pre-included in the HTML document, since many contents are dynamically generated such as ads inserted by third-party JavaScript. The extension stays at each site homepage for 45 seconds before switching to next site<sup>2</sup>. After collecting all the homepage logs, we run a toolkit mainly written in Java to retrieve image object information from each log based on extension type and to decide whether an image object is hosted locally or hotlinked from other sites based on URI.

### 2.2.1.3 Data Analysis

We first describe the analysis and processing of image measurement results, which are summarized in Table 2.2. For the homepage of each website, it may include two types of images: those hosted by the site itself and those hotlinked from other sites. We can distinguish these two types of images by comparing URIs of the site and images. If the homepage of a site links any images hosted by other sites, the site is categorized as *Site with Hotlinking Behavior* (Column 2 in Table 2.2). There is nothing wrong with using hotlinking. However, the key issue here is to differentiate between authorized and unauthorized hotlinking. Our large-scale measurement involves 1,453 sites, and the majority of these sites link images from many other sites. It is very time-consuming to manually

---

<sup>2</sup>We measure the load time of 100 sample site homepages. Most of them do not exceed 20 seconds. Thus, we set a safe threshold value here, 45 seconds, which is long enough for fully displaying a page.

**Table 2.2: Image Hotlinking Distribution per Site Homepage**

Category	Sites w/ HL Behavior	Sites w/ Unauthorized HL	Unauth-HL Image Ave.	Sites w/ Authorized HL	Auth-HL Image Ave.
Arts	70	19 (27.1%)	2.79	69 (98.6%)	10.55
Business	36	5 (13.9%)	1.60	34 (94.4%)	14.41
Computers	40	7 (17.5%)	2.29	39 (97.5%)	16.15
Games	57	11 (19.3%)	3.00	56 (98.2%)	11.45
Health	50	6 (12.0%)	2.00	49 (98.0%)	10.47
Home	64	18 (28.1%)	1.78	61 (95.3%)	11.10
News	78	16 (20.5%)	1.88	78 (100%)	13.22
Recreation	39	8 (20.5%)	3.38	39 (100%)	9.54
Reference	35	6 (17.1%)	3.17	34 (97.1%)	11.74
Regional	26	8 (30.8%)	1.50	25 (96.2%)	19.32
Science	29	13 (44.8%)	2.15	26 (89.7%)	7.19
Shopping	56	23 (41.1%)	3.22	52 (92.9%)	11.96
Society	35	13 (37.1%)	3.46	28 (80.0%)	13.82
Sports	41	18 (43.9%)	3.72	36 (87.8%)	8.31
World	39	12 (30.8%)	2.50	31 (79.5%)	10.48
Blogging	395	273 (69.1%)	9.00	360 (91.1%)	32.35
Total w/o blog	695	183 (26.3%)	2.55 (avg)	657 (94.5%)	11.98 (avg)
Total	1090	456 (41.8%)	2.96 (avg)	1017 (93.3%)	13.25 (avg)

check whether a site obtains the authorization from another to hotlink an image. Based on our observations, we apply the following three rules for processing the measurement results.

- Rule 1. If the page of site A contains any iFrames or scripts from site B, and B includes images in iFrames or dynamically inserts via scripts, it is considered that A is authorized to hotlink those images from B. This fact implies the cooperation relationship between the two sites. In this case, B plays the role of content (image) provider. Take online ad assignment as an example. A includes an iFrame from syndicator B, and B puts links of ad images in the iFrame, and dynamically changes them to update ad content. This rule also applies to many other web applications, such as site traffic monitoring, and webpage tagging.
- Rule 2. A white list is created to cover the popular sites in the categories which generally authorize hotlinking from themselves. The representative categories include advertising syndicators (such as Google Syndication [32], 2mdn [1] and DoubleClick [24]), web performance

accelerators (such as Akamai and Speedera [3]<sup>3</sup>), and image hosting/cloud service (such as Flickr [25] and CacheFly [21]<sup>4</sup>). Moreover, the white list includes a few partnerships between specific sites, such as yahoo.com and yimg.com, where the latter hosts images for the former. This partnership suggests authorized hotlinking.

- Rule 3. Web pages often include a large amount of very small images mostly in the format of GIF and PNG. These small images are mainly used as tags of social networks (such as Facebook, Dig and Twitter), toolbar logos, face expression symbols, and so on. Generally hosting sites of such small images authorize and solicit hotlinking to spread their brand names. In our measurement, we set the threshold as 10KB<sup>5</sup>. Any images whose sizes are less than the threshold are classified as authorized hotlinking.

A program equipped with the above rules is used to automatically distinguish unauthorized hotlinking behaviors from authorized ones. If an image follows any of the rules, it is labeled as an *authorized hotlinked image*. Otherwise, it is labeled as an *unauthorized hotlinked image*. The automatic determination saves us from burdensome manual check over hotlinked images. However, the side-effect is the possible false negatives (unauthorized hotlinking misjudged as authorized hotlinking) and false positives (authorized hotlinking misjudged as unauthorized hotlinking). We randomly select 20% of the sites with hotlinking behavior to perform the manual check. For each hotlinked image, we examine the characteristics of hotlinking and victim sites, their relationship, and the content and size of the image. Then we determine whether hotlinking the image is authorized or not, and compare with the decision made by the program. The false negative ratio is 0.1%. Very

---

<sup>3</sup>These sites let normal sites to hotlink special-purposed images, and trace image requests to gather statistical information about normal sites

<sup>4</sup>These sites provide online storage services most of which are free, and their policies explicitly allow hotlinking

<sup>5</sup>According to the measurement results in our crawl experiment, the majority of this type of images are smaller than 10KB.

few images should have been judged as unauthorized hotlinking based on their contents. However, due to their sizes smaller than the threshold value in Rule 3 (namely 10KB), they are misjudged as authorized hotlinking by the program. The false positive ratio is 3.7%, which is caused by the following two reasons.

- The incompleteness of the white list in Rule 2. For example, it does not contain a site (imagevenue.com) providing free image hosting. Hotlinking images from this site should be considered as authorized. Besides, the list misses specific relationships between some sites. For example, our manual check discovers the redirection from bdd.com to randomhouse.com, which suggests the site partnership. Thus hotlinking is authorized between the two sites.
- The setting of image size threshold in Rule 3. Our manual check observes that some images whose sizes are greater than the 10KB threshold. However, their contents of ads strongly suggest the authorized hotlinking.

Our manual check confirms the accuracy of the automatic classification of authorized and unauthorized hotlinking. Now we analyze the statistical data listed in Table 2.2. A site containing such images is categorized as a *site with authorized hotlinking behavior* (Column 5 in Table 2.2). We divide the sum of authorized hotlinked images by the number of sites with authorized hotlinking behavior to obtain the average of authorized hotlinked images per site (Column 6 in Table 2.2). Similarly, we count the *sites with unauthorized hotlinking behavior* (Column 3 in Table 2.2) and compute the average of unauthorized hotlinked images per site (Column 4 in Table 2.2), respectively. If a site hotlinks both types of images, it is counted in both Column 3 and Column 5.

In the first 15 categories without blogging, 46.3% of the sites (namely, 695 out of 1,500) hotlink images. In other words, more than half of the sites (53.7%) host all the images by themselves

and do not hotlink. Among the hotlinking sites, most sites (94.5%) manifest authorized hotlinking behaviors, and the average of authorized hotlinked images per site is 11.98. Only a minority of sites (26.3%) have unauthorized hotlinking behaviors, and the average of unauthorized hotlinked images per site is 2.55. Such measurement results are expected. Those top sites from the 15 categories are mostly owned by large organizations. They have adequate online storage space and traffic quota, and can host any images by themselves. They do not have strong motivations to "steal" images from others. They perform hotlinking mainly for site interaction. We manually check the nature of authorized hotlinked images, and find out that most of them are for online advertising, webpage tagging, and site partnership displaying.

However, the blogging category is an exception. Most blogging sites (395 out of 400) hotlink images. Among them, 273 blogging sites have unauthorized hotlinking behaviors. The unauthorized ratio of 69.1% is much higher than the average ratio (26.3%) of the other 15 categories. The average of unauthorized hotlinked images per site is 9.00, much higher than the average (2.55) of the other 15 categories. It is not surprising to observe that hotlinking is severe in the blogging category. The reasons are: (1) many blog sites are operated by individuals who are not professional web developers. They tend to use search engines to look for pictures and then directly link them instead of hosting them. Besides laziness, they do not have the clear conscience on copyright infringement their (unauthorized) behaviors have done. (2) many blog sites do not have enough online storage and traffic quota, and have to intentionally hotlink from other sites. On the other hand, the proportion of blogging sites that have authorized hotlinking behavior is 91.1%, which is very close to the average (94.5%) of the other 15 categories. This is because blog sites also have needs of regular site interaction, such as online advertising and traffic monitoring. However, the average of authorized hotlinked images per site is 32.35, much higher than the average (11.98) of the other 15 categories.

Blog sites link a large amount of small images for user avatars and face expressions. Furthermore, blog pages display more ad images than regular webpages.

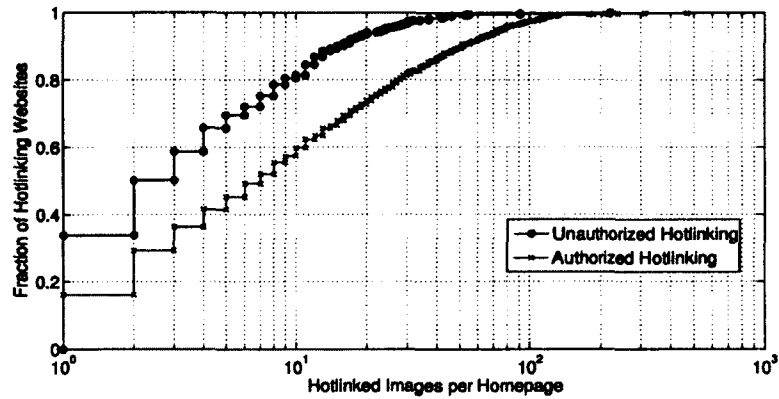


Figure 2.5: CDF of the number of hotlinked images per homepage

For those sites with unauthorized hotlinking behaviors (456 in our measurements), Figure 2.5 shows the cumulative distribution function (CDF) of the number of unauthorized images hotlinked per homepage. Around 81.2% of the site homepages hotlink at least 10 images. While the maximum value of outdegree is 221, the mean is 6.52, and the standard deviation is 13.36. The fact suggests that, if a site conducts unauthorized hotlinking, the behavior may be regular instead of occasional. The corresponding CDF curve of authorized hotlinking is also shown in Figure 2.5 as a reference. The maximum value of outdegree in the authorized curve is 469, the mean is 18.63, and the standard deviation is 31.90.

Table 2.3: Unique Victim Site Distribution by TLD (16 categories)

Top-Level Domain	No. of Unique Victim Sites
com	2,453 (78.8%)
cc	343 (11.0%)
net	161 (5.2%)
org	118 (3.8%)
gov	6 (0.2%)
misc	31 (0.9%)
Total	3,112 (100%)

Since a great amount of images are hotlinked all over the Internet, besides "hotlinking culprit", we also want to know the general distribution of victim sites. For images hotlinked without authorization in our measurement, we get URLs of their hosting sites. Table 2.3 shows Top-Level Domain (TLD) distribution of victim sites. The majority of victims belong to the .com domain (78.8%). The second largest victim domain is the .cc domain (11.0%). The remaining domains account for a small portion (10.2%).

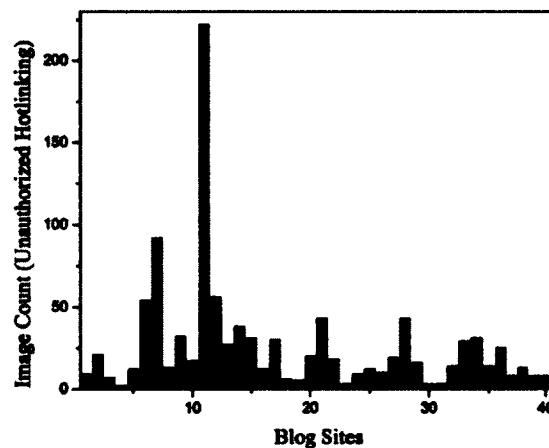


Figure 2.6: Count of Hotlinked Image without Authorization for top 40 Blog Sites

Since image hotlinking is severe in the blogging category, we conduct a deep measurement which targets images hotlinked by the site, not merely by the homepage. More specifically, we pick up top 10% sites (namely, 40 out of 400) in the blogging category based on the number of hotlinked images per homepage. We modify wget [29] to crawl these top 40 sites. For each site, starting from the homepage, the crawler visits the internal pages linked in the homepage in a recursive way, with the recursive depth set as three<sup>6</sup>. After acquiring an (incomplete) internal page list of a site, we still use our Firefox extension to visit all these pages to accumulatively log images hotlinked by the site.

<sup>6</sup>We believe that the 3-depth-level is a good balance between site coverage and measurement time. A larger depth will incur much greater running time in a nearly exponential way and also generate more duplicate pages.

Figure 2.6 shows the total number of hotlinked images for the top 40 blogging sites we measured. A site surprisingly hotlinks 222 images. Averagely speaking, each of these 40 blog sites hotlinks about 25 images. It confirms again that these target blog sites do hotlink a large amount of images without authorization.

Based on the above measurement results, we can draw the following conclusions. (1) The behavior of hotlinking images is very common over the Internet. (2) For the majority of top sites in the first 15 categories except blogging measured by us, they have their own servers for image storage. Linking images hosted by other sites is usually for web interaction (such as advertising and traffic tracing). Such behaviors are authorized by hosting sites. (3) A small proportion of sites (such as blog sites measured above) hotlink a large amount of images without authorization. This unethical behavior infringes the interest and rights of hosting sites.

### **2.2.2 Measurement of Hotlinked Software Packages**

Among the various types of files hotlinked over the Internet, the software installation packages (such as .exe and .rpm) are more prone to becoming hotlinked. Due to the large file size, hotlinking may cause significant resource consumption to hosting sites. Therefore, software package is selected as the measurement target in the file category of hotlinking. In the previous image-centric measurements, we start from hotlinking sites and trace back to victim sites. In this part, we reverse the order and trace from victims to hotlinkers. The measurement procedure is explained as follows.

Based on download times and popularity, we choose 100 top free software packages as suggested by [26, 16]. Our software set covers various categories ranging from security software to developer tools. For each software package, we manually obtain its official download URL that usually belongs to its author/publisher site. We use a PHP-written script to search web pages that



contain official download URLs via the standard query API provided by Google. We argue that, including the official download URL on a third-party page is a kind of hotlinking, because the visitor can directly download the software by clicking on the hotlinked URL without visiting the official download page. In most cases, such hotlinking behaviors are not authorized or expected by software owners. Software owners may use CDN (content delivery network) sites to spread files for faster download. However, this case is not hotlinking at all, since CDN sites do host software packages by themselves.

For each package, we generate a log including the first 100 results (namely, URLs of hotlinking pages) returned by Google. The result ranking is decided by search relevance and popularity. Thus, we believe that the top 100 results are appropriate for the measurement sampling purpose. If the number of the results returned is less than 100, we take the actual number. After collecting logs of these 100 software packages, we use our Java-written toolkit to process them and do a comprehensive analysis presented as follows.

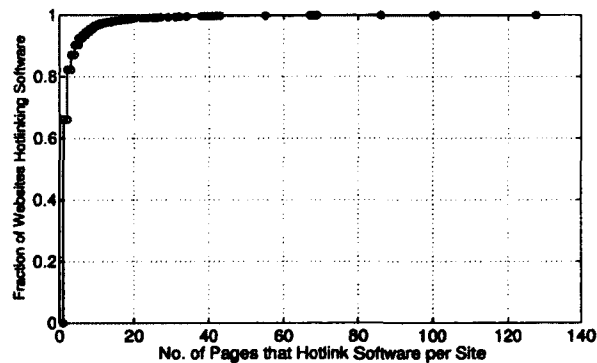


Figure 2.7: CDF of the number of hotlinked software packages per site

In our measurement result set, the 100 software packages are hotlinked by 3,020 sites in 7,539 unique web pages. For the above hotlinking sites, Figure 2.7 shows the CDF of the number of

hotlinked software packages per site. Around 33.8% of the sites hotlink 2 or more software packages (may in different pages). The maximum value of the CDF curve is 128. It is a forum-style site where "warmhearted" users publish 128 hotlinking download URLs (including duplicates) in multiple posts. The curve mean is 2.50 with the standard deviation of 5.79.

**Table 2.4: TLD Distribution of Unique Sites that Hotlinking Software Packages**

Top-Level Domain	No of Unique Hotlinking Sites
com	1859 (61.6%)
cc	481 (15.9%)
net	372 (12.3%)
org	202 (6.7%)
info	42 (1.4%)
misc	63 (2.1%)
Total	3020 (100%)

Moreover, Table 2.4 lists the distribution of the 3,020 hotlinking sites by Top-Level Domain. The majority are from the .com domain (61.6%). Subsequently, the .cc and .net domains contribute 15.9% and 12.3%, respectively. The remaining domains account for 10.2%.

The above analysis manifests that hotlinking software packages is a common problem over the Internet. It also happens to other types of file resources, such as documents and audio/video clips. Because of the large file size, frequent hotlinking incurs significant consumption in network bandwidth and computing resources. It is essential to deploy anti-hotlinking methods to protect hosting sites.

### **2.2.3 Postmortem Analysis of A Hotlinking Attack**

To fully understand the damages caused by hotlinking towards hosting sites, such as system burden and traffic theft, we collected raw traces from a victim server and performed a forensic-style postmortem analysis on a real hotlinking attack.

The hotlinking attack is briefly described as follows. One user of the victim server hosted a folder of many images of a popular commercial product under his web directory. He posted a lot of articles containing some of the above images at a few third-party sites. Due to the attractive contents and images, those articles were frequently referred by others. Eventually they drew a large amount of click traffic. Since the images were hosted at the victim server, numerous image requests were redirected to it, consuming many computing and network resources. Finally, the victim server was overwhelmed and crashed. The user was not conscious of his hotlinking behavior and the serious damages induced to the server. It is the system administrator who noticed the crash of the victim server and blocked the public Internet access to that user's directory, which ends the hotlinking event lasting in a continuous period of 40 days from the first week December, 2008 to the early January, 2009.

The victim server is installed with Apache 2.2.4 running on an Intel Xeon 64-bit workstation, which is equipped with quad-processors of 3GHz, 12MB L2 cache, 16GB memory, 300GB hard disk and 1Gbps LAN connection. The victim server grants the direct HTTP access to any files in a user's web directory if the file path is given correctly. Namely, hotlinking is allowed, and no anti-hotlinking defense is deployed. The image folder contains around 1,000 images in the format of JPEG. The size of a single image varies from 50KB to 300KB, and the total size of the image folder is about 130MB. Each entry in the raw server log records the response to an HTTP request, including fields like the client IP address, timestamp, the file path of the requested object at the server, and the Referer of the HTTP request. The total size of the 40 days' logs is 400MB. After removing those entries irrelevant to image hotlinking, we reduce the total log size to around 200MB.

With the help of the Referer field, we trace back to the hotlinking source. A set of articles were posted at some third-party sites, more specifically, nine BBS-style forums and one EBay-style

shopping site. The article pages included images hosted on the victim server in the form of ``. In this way the hotlinking relationship forms between those third-party sites and the victim server. Most of those sites have a large user population, and generate a large trace of user requests. The number of images embedded in the article pages ranges from 5 to 80. One page hotlinks 80 images with the total size of 12MB. When the browser displays the page, it sends an HTTP request for each embedded image to the victim server. The server returns the image.

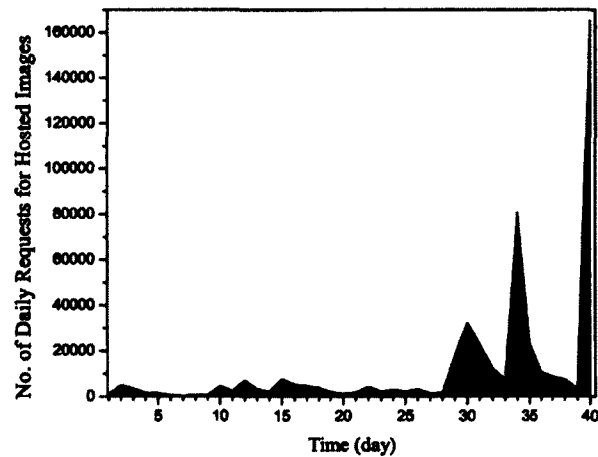


Figure 2.8: Daily traffic in terms of image requests

Figure 2.8 shows the number of daily requests for hosted images. The average number of daily requests over the 40-day window is 11,872. The daily number gradually increases, and reaches the summit of 165,430 on the last day when the victim server was crashed. The curve development can be plausibly explained as follows. With the increase of user views and replies, the rank of the article boosts. As a case of Matthew Effect (i.e., the rich get richer and the poor get poorer), it draws more users, bringing more click traffic. There are three spikes over the last 10 days. We manually

checked the posts during that period, and found the obvious evidence of scripts<sup>7</sup> that automatically post replies to articles to keep them staying in the first few pages of the forums, thus attracting many more viewers.

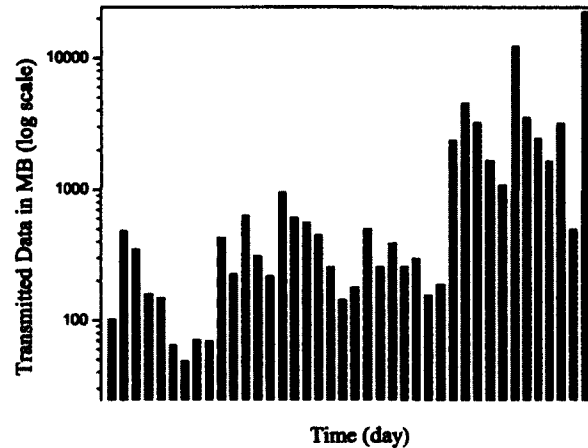


Figure 2.9: Daily Data Transmission (in MB) Caused by Hotlinking

Figure 2.9 shows the daily data transmission caused by hotlinking. We first compute the data transmission,  $T_i$ , caused by a single file, as  $T_i = F_i * R_i$ , where  $F_i$  is the size of the file  $i$ , and  $R_i$  is the number of requests for the file  $i$ . The daily data transmission is calculated as the sum of that of all the files requested that day, i.e.,  $T_d = \sum_i T_i$ . The curve in Figure 2.9 is similar to that in Figure 2.8. The data transmission increases gradually over the first 30 days, and surges drastically over the last 10 days. The average daily data transmission caused by image hotlinking is around 1.7GB. The maximum occurs on Day 40 at 22.7GB, followed by Day 35 at 12.3GB.

Figure 2.10 illustrates the number of daily client IP addresses, which can be used to roughly estimate the number of daily visitors. The average of daily IP addresses is 490, while the maximum of 2,215 appears on Day 35 followed by the second highest of 1,950 on Day 40. Over the 40 days,

<sup>7</sup>For example, the reply interval is short and almost fixed. Furthermore, many recently registered accounts are used.

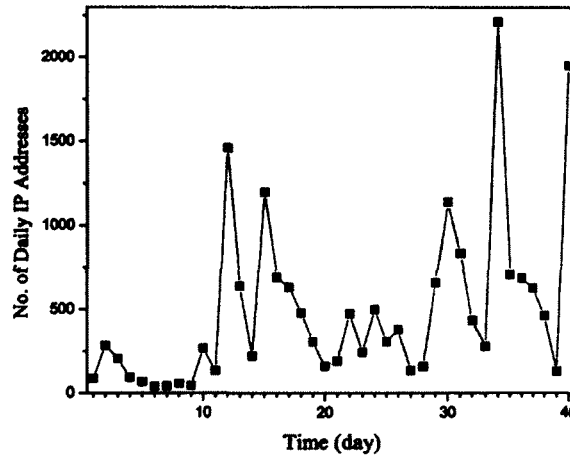


Figure 2.10: Daily traffic in terms of client IP addresses

on average, each IP address requested 24 hotlinked images.

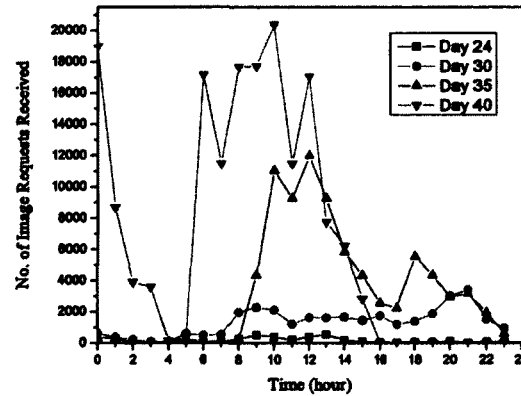


Figure 2.11: The hourly traffic distribution of four selected days

Among the 40-day logs, we choose four days for more detailed analysis. The result is presented in Figure 2.11. Days 30, 35, and 40 are selected due to their top data transmissions. The number of the daily IP addresses of Day 24 (497) is the closest to the average (490), and thus that day is used as the average case. The curve shapes of the first three days are similar. The peak appears from 9:00 to 15:00, followed by another wave from 17:00 to 21:00. This matches the regular timetable of human

visitors, since more people tend to surf the Internet during working hours and in the evening. Day 40 is a special case, where starting from 8:00, the hourly traffic reaches 18,000 requests with the summit at 20,000 (namely 5.6 requests per second). The victim server is overwhelmed by the large amount of incoming HTTP requests. In the following period, its performance drops sharply and the response becomes extremely slow. The server crashes around 16:00. The system administrator restarts the server manually and blocks the public access to the user image folder afterwards.

## 2.3 Framework Design

Based on the existing network security techniques, we present an anti-hotlinking web framework for hosting sites. Our design goal is to greatly increase the hotlinking difficulty and to defeat most common forms of hotlinking with easy deployment. In reality, webmasters can control the granularity of anti-hotlinking by defining different protection policies and applying them to the framework based on the requirements of their sites. Currently the framework provides two policies, *Strict Policy* and *Loose Policy*, for the demonstration purpose. In this section, we first describe the design details of the framework and then present the two protection *policies*.

### 2.3.1 Design Details and Modules

Figure 2.12 illustrates the anti-hotlinking framework that consists of three major modules. The *HTTP Request Filtering Module* filters incoming HTTP requests and blocks direct access to hosted resources. The file entrance page contains the other two modules. The *Session Creation/Authentication Module* creates and manages sessions to maintain the HTTP communication status between the server and client. Different protection policies may require different steps, and the user must com-

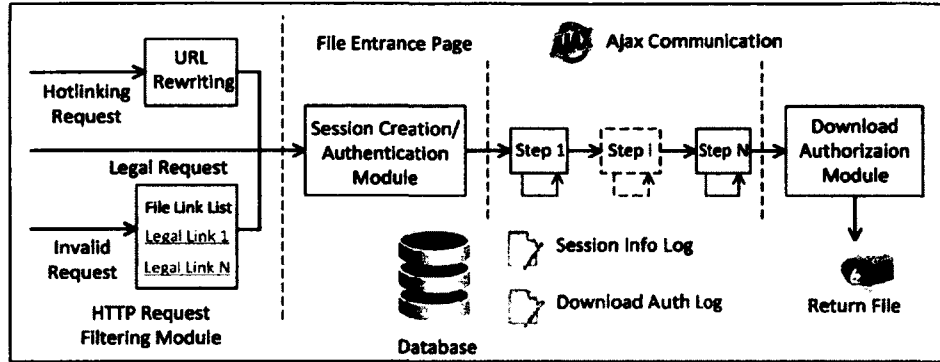


Figure 2.12: Anti-Hotlinking Framework Overview

plete them to become eligible for downloading. The *Download Authorization Module* checks the download authorization log and determines whether the download request can be granted. If so, the server will return the file to the client. The intra-page communication of the file entrance page is supported by the AJAX techniques. The page address in the browser does not change, and only the page content in the inner window updates smoothly. The detailed description of these modules are given as follows.

### 2.3.1.1 HTTP Request Filtering Module

The main function of this module is to transform the incoming HTTP request into the legal form required by the framework and to block the direct access to hosted resources. The module uses the following three functional blocks.

Table 2.5: File Storage Log

Unique File_ID	File Name	Storage Path
T85X4PNS	install.rpm	root/files/

**Unique file ID and entrance page.** The site assigns each of its hosting files a unique file ID, and this solves the problem of duplicate file names. Currently, our framework uses three types of



information to generate the unique file ID: original file name, file upload timestamp and user ID (either the user account name or the IP address) where appropriate. The site then hashes the above information into the file ID and maintains a file storage structure table as shown in Table 2.5. A unique entrance page can be created for each file based on its file ID.

**Limiting HTTP requests.** Most sites without anti-hotlinking defense directly grant HTTP requests for hosted resources as long as the resource path is given correctly. Limiting HTTP requests helps the site block the direct access to hosted materials. As an option, the site can use URL rewriting of HTTP requests to achieve it. Suppose now the framework only directly grants HTTP requests for web pages (whose extensions are .htm and .php in our prototype) and some accessory web objects required for normal page display (such as background images), while requests for all other types of objects are redirected to corresponding entrance pages based on unique file ID. For example, the direct HTTP request for the file `install.rpm` in the form of `www.site.com/files/install.rpm` is prohibited and rewritten into `www.site.com/download.php?fid=T85X4PNS`.

**Handling different HTTP requests.** The module may receive three types of HTTP requests for a file, and finally it transforms them into Legal Request, the only request type accepted by the framework. The three types of HTTP requests are listed as follows. (1) *Legal Request* (in the form of `www.site.com/download.php?fid=T85X4PNS`, with a valid fid) is directly brought into the file entrance page. This format is always used by all internal pages of the hosting site. (2) *Hotlinking Request* (in the form of `www.site.com/files/install.rpm`, without a fid) that directly requests files is rewritten into Legal Request based on the unique file ID mapping and then redirected to the file entrance page. (3) *Invalid Request* (in the form of `www.site.com/download.php?fid=§85X4PNS`, with an invalid fid) is redirected to a file-list page that shows legal download links of hosted files on the server. When a user clicks on a legal download link, it generates a Legal Request and the user

can continue to download.

### 2.3.1.2 Session Creation/Authentication Module

The anti-hotlinking framework needs to maintain the interaction status between the client and server to determine whether the client becomes qualified to download the requested file. Both cookie and session mechanisms can be used to maintain HTTP communication states. Our framework chooses to use the session mechanism. The main reason is that, the session mechanism only has to store session ID on the client-side, and maintains most session information on the server-side. It greatly reduces the risk of the client-side forging and hacking. Session ID can be stored on the client-side in the form of cookie or URL parameter. In either way, the server must authenticate the received session ID to make sure it is not only valid, but also originally assigned to the client. Because HTTP sessions build upon TCP connections, this module uses the essential information extracted from the underlying TCP connection along with browser signature to perform session ID authentication. The TCP information mainly includes the IP addresses and port numbers of source and destination. For every session it initiates, the server creates an entry in the form of [session ID, TCP info, client browser signature] in the *Session Info Log*.

When the HTTP request arrives at the file entrance page, the module determines whether the client has an existing valid session by authenticating the session ID. The authentication procedure is shown as Figure 2.13. If the authentication succeeds, the module creates an entry in the *Download Authorization Log* and the client can start to execute the required download steps. Requirements specified by different protection policies may vary. We give an example set of steps in Section 2.3.2. After the steps are fulfilled, *Download Authorization Module* decides whether the download request will be granted or not.

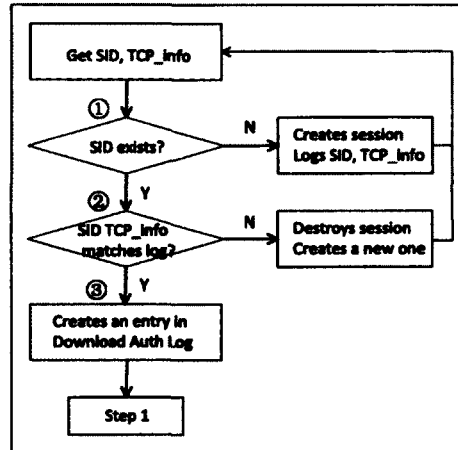


Figure 2.13: Session Creation and Authentication

### 2.3.1.3 Download Authorization Module

Table 2.6: Download Authorization Log

SID	FID	Step_1	Step_i	Step_N	Auth	Auth_code	Expire_TS
5k0642	T85X4PNS	True	True	True	True	d383e3	1227213501

The *Download Authorization Log* is a core log that stores information used by the site to decide whether the file request can be granted or not. Table 2.6 lists a simplified log entry. *Session\_ID* is associated with the client-side information (mainly about TCP connection and browser signature), and *File\_ID* is unique to each file. The first two fields together show a specific client requests to download a specific file. The *Step\_i* field is set as true after the corresponding step is completed. After all the required steps are finished, the last three fields are set as follows. The *Authorization* field is turn into true, the *Authorization Code* field is filled with a random string, and the *Expire Timestamp* field carries a timestamp that specifies when the *Authorization Code* expires. The *Download Authorization Log* uses *Session\_ID* and *File\_ID* to locate the corresponding entry in the *Download Authorization Log*. If (1) the *Authorization* field is true, (2) the *Authorization Code* field is valid, and (3) the *Expire Timestamp* field is greater than the current timestamp, then the module authorizes

the download request.

The module uses `File_ID` to locate the associated file entry in the *File Storage Log*, and reads the file via the full file path. The site constructs an HTTP response containing the requested file and returns to the client. The framework provides the following two options to handle the *Authorization Code* field after the file request is granted. The first is the *One-time Code Use* option. Namely the code may only be used once and expires after then. If the same user wants to download the same file again, he will be redirected to Step\_1 to start over even though the session is still alive. The second is the *Repeated Code Use* option. The code can be repeatedly used until it expires. This option enables the user to download the same file multiple times during a certain time window. To prevent the code from being abused, the web server can limit both the maximum number of simultaneous download connections per IP address and the download speed per process.

### 2.3.2 Strict Policy

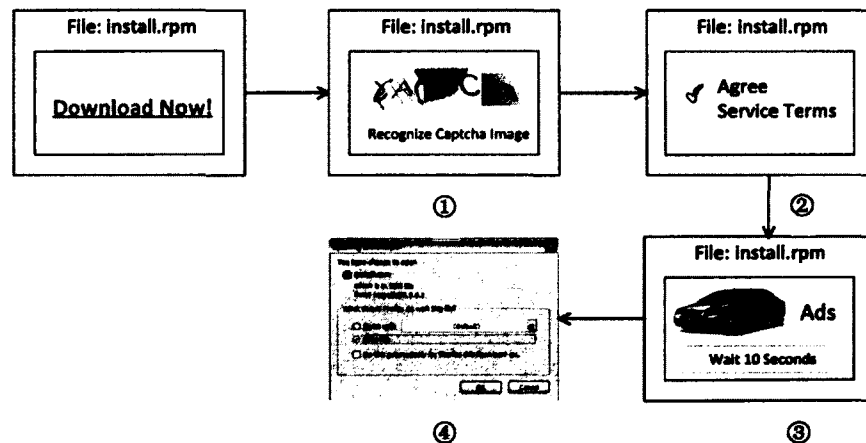


Figure 2.14: User Download Procedure

Hosting sites often accommodate high-importance and large-size resources, such as software

packages and documents. Hotlinking such resources may cause serious damages to hosting sites. We introduce *Strict Policy* to protect this type of resources as follows.

The *HTTP Request Filtering Module* filters the incoming resource request into the form of Legal Request. When the request is directed to the *File Entrance Page*, the *Session Creation/Authentication Module* guarantees an authentic session between the server and client.

The key feature of *Strict Policy* is that, it specifies a set of steps the user must fulfill to become eligible for downloading. We emphasize that, the policy can generate as many steps as needed and customize requirements for each step. These steps can serve different goals for hosting sites in reality, like distinguishing human users from machines, displaying sponsors ads, and encouraging users to buy premium download accounts, etc. We leave up to the webmaster the details of designing steps. Our framework implements a combination of three simple steps as shown as Figure 2.14 purely for the demonstration purpose. The user must (1) correctly recognize a CAPTCHA image, which distinguishes human users from machines<sup>8</sup>, (2) check the Terms of Service checkbox, and (3) wait a short time to activate the download link. The *Strict Policy* sets the download authorization code to be one-time use. If the user wants to download the same file again, it has to repeat the steps to generate another authorization code.

### 2.3.3 Loose Policy

While the *Strict Policy* is stringent and suitable for protecting the resources of great importance, it is too cumbersome for users to download these widely-used web objects such as images. As an alternative, we present a light-weight policy, *Loose Policy*, for protecting common web objects of

---

<sup>8</sup>Using CAPTCHA is an option of distinguishing human users from machines. Its effectiveness against automated clicks is out of the scope of the chapter.

less importance. Its design principle is to strike a good balance between resource protection and anti-hotlinking cost.

Suppose hosted objects (such as images) are included in some web pages of the hosting site. After a browser loads and parses the webpage, it sends a request for the embedded object to the hosting site. According to Figure 2.12, the server rewrites the incoming HTTP request into Legal Request (like `www.site.com/loose_download.php?fid=57OUK14N`). The download control page (`loose_download.php`) calls the *Session Creation/Authentication Module* to judge whether the browser has an active session with the server or not. If so, the *Download Authorization Module* will return the object to the browser. Note that the process of executing steps is turn off in the Loose Policy. The whole download control procedure does not involve any user interaction, and is totally transparent to users. This is a big difference from the Strict Policy. Furthermore, the *Download Authorization Module* takes the option of *Repeated Code Use*. This allows the browser for multiple downloads towards the same object during the session lifetime.

If there is no active session between the browser and hosting site, it is very likely the direct object request is triggered by a hotlinking webpage<sup>9</sup>. The control page will redirect the browser to a default page that builds an active session and displays some notification information like *The requested object is originally hosted at www.site.com (click here to visit the genuine hosting site)*. The page stays for a short time period (like three seconds) customizable by the site. After that, the request will be granted and object will be sent to the browser to appear in the page.

---

<sup>9</sup>It is almost impossible for a user to know the object URL that is long and contains random strings without visiting the hosting page. There is a slim chance that the user enters the object URL on the browser to directly visit it. In this situation, we think that the user intends to visit the hosting site. The redirection action will happen, but it does not hinder the user's browsing experience.

## 2.4 Implementation

In this section, we describe the implementation details of our anti-hotlinking framework prototype. It is deployed on the server-side and does not require modifications on the client-side. It only requires browsers to run a small amount of assistant JavaScript codes that are supported by modern out-of-box browsers.

### 2.4.1 Web Server Setup

We choose Apache (version 2.2.8) [7] as the web server based on which our anti-hotlinking prototype is deployed. It runs at a workstation with Intel dual-processor 2.2GHz and 2GB of RAM. Three additional modules, `mod_limitipconn`, `mod_bandwidth` and `mod_rewrite`, are added to the server. The `mod_limitipconn` module [9] limits the maximum number of simultaneous download connections per IP address. Similarly, the `mod_bandwidth` module [8] limits the download speed of a single connection. The `mod_rewrite` module [10] rewrites the requested URL on-the-fly based on configuration rules with the style of Perl formal expressions. In our framework, rewriting HTTP requests is achieved by modifying `.htaccess` file and putting it into proper directories.

### 2.4.2 Technical Details

We use PHP 5.2.5 [40] as the back-end programming language. Most web pages in our framework are written in PHP. Our framework uses PHP functions to implement session communication, and to store related variables in the `$_SESSION` array. `Session_ID` can store on the browser in the form of cookie. If the browser disables cookies, PHP can detect it and transfer `Session_ID` to the browser as a query parameter of URIs. The webmaster can enable the PHP Transparent Session Support function by turning on the statement `session.use_tran_sid = 1` in the `php.ini` configuration file. In

the current PHP page, `Session.ID` will automatically append to all the internal URIs of the hosting domain. Another important function module, CAPTCHA [22], is also implemented in PHP. We use a pseudo-random algorithm to generate a string consisting of a fixed number of random characters. They are drawn on the image in a random font with the help of the GD library [27]. Some random background noise is also added to thwart character recognition algorithms.

Our framework uses a small amount of JavaScript codes that are embedded in webpages and run at the client-side. It provides a few additional functions to enhance the user experience, and does not affect the core download control procedure of the framework at all. If the browser disables running JavaScript, a client can still finish the download procedure. The framework provides Ajax-style (Asynchronous JavaScript and Xml) [2] intra-page communication based on `prototype.js` [41]. The advantage brought by Ajax is that, it retrieves data from the server asynchronously in the background without interfering with the display of the current page.

## **2.5 Evaluation**

In this section, we present the system evaluation of the proposed framework in terms of security and usability, with the focus on the Strict Policy.

### **2.5.1 Security Analysis**

From the security perspective, the Strict Policy provides a much stronger defense against hotlinking attacks than the Loose Policy. We use the Strict Policy to demonstrate the effectiveness of our anti-hotlinking framework against the four different types of hotlinking attacks presented in Section 2.1.1, respectively.



### **2.5.1.1 Effectiveness against Direct Hotlinking**

Since the site applies URL Rewriting towards incoming HTTP requests, the request for a file will be redirected to the download entrance page. The server does not directly return the file in response to the HTTP request. Therefore, the attack of Direct Hotlinking will be defeated.

### **2.5.1.2 Effectiveness against Hotlinking via Referer Fabrication**

The site does not use the HTTP\_REFERER field in the received HTTP request header to determine whether it should return the file or not. Instead, it uses the control procedure shown in Figure 2.12 to make the decision. Thus, fabricating an HTTP\_REFERER by using the domain of the hosting server does not help hotlinking at all.

### **2.5.1.3 Effectiveness against Hotlinking via Cookie Vulnerabilities**

To demonstrate the effectiveness against a hotlinking attack that explores cookie vulnerabilities, we suppose Session\_ID is stored on the browser in the form of cookie. The hotlinking page, H.com/h.htm, can contain an iFrame from the victim site V.com to make the browser establish a session with V.com. When the browser requests a file that is hotlinked in h.htm and hosted by V.com, V.com will call the *Session Creation/Authentication Module* to validate the current session. The session validation will pass. However, the hotlinking will fail to execute the required steps (shown in Figure 2.12). The reason is that, the same-origin policy prevents the malicious code on h.htm to access the iFrame of V.com. Therefore, the malicious code cannot complete the required steps, such as CAPTCHA image recognition, even if it has the ability of automated scripting. Since the hotlinking page disables the user to view the legal page by setting the iFrame hidden, the user will not help finish the required steps either.

```
  
<a id="fileLink" href="www.V.com/files/download.php?fid=T85X4PNS&sid=5k32d0"  
>Download RMP Package</a>
```

Figure 2.15: Modified snippet exploring session vulnerability

#### 2.5.1.4 Effectiveness against Hotlinking via Session Vulnerabilities

Suppose the malicious code shown in Figure 2.4 manages to steal the session ID, and append it to the file download hotlink. The last two HTML statements in Figure 2.4 can be modified as Figure 2.15 shows. Now the legal file request along with the valid session ID and file ID is sent to the web server. It can pass the check of *Session Creation/Authentication Module*. However, the *File Entrance Page* (namely, `download.php`) cannot be displayed on the client side. It is discarded by the browser because the returned file type is HTML document rather than image expected by the `<img>` tag. The download steps on this page cannot be completed by the user. Thus, this type of hotlinking attacks will also be defeated.

### 2.5.2 Usability Analysis

After the Strict Policy is enforced on the hosting site, the download request for the hotlinked file will be redirected to the specific entrance webpage. First, the site creates a session with the browser or validates the existing session. This procedure is totally transparent to the user, and the induced time delay is hardly to be noticed. Second, the user needs to fulfill some steps required by the hosting site before he becomes qualified for downloading. The time consumption of this procedure is determined by the details of the requirements. In our prototype, the three example steps take around 20 seconds to complete<sup>10</sup>. On one hand, this is unavoidable trade-off a hosting site has to

---

<sup>10</sup>The third step requires the user to wait 10 seconds, which can be skipped by the webmaster in reality.

make; on the other hand, the hosting site must design the download steps carefully to maintain the users' interests of the site.

The Loose Policy is responsible of delivering the requested object (often images) to the client browser. It does not involve any user interaction. When the server receives an object request, the Loose Policy performs additional steps listed as follows. In comparison with the direct HTTP response, it causes some minor delays.

- If there is no active session, the server needs to create a session for the client, and logs related information to the session file. The time consumption is labeled as Delay\_1.
- If the session ID rendered by the client does not match the server log (i.e., it is stolen from others), the server will delete the current session and create a new one. The time consumption is labeled as Delay\_2.
- If the session ID is legal, it will pass the server authentication. The time consumption is labeled as Delay\_3.

To measure the delay in each of the above cases, we use a Firefox browser to visit a Apache server protected by the Loose Policy. We conduct 50 visits for each case. The server records the timestamps (in millisecond) of the event start and end, and calculates the delay. On average, Delay\_1 is 10.0ms, Delay\_2 is 7.3ms, and Delay\_3 is 11.1ms. Overall, the additional delays induced by the Loose Policy are minor and hard to be noticed by the client.

## 2.6 Related Work

We have already discussed several common hotlinking attacks in Section 2.1, and in this section we survey related work in a broader scope.

Hotlinking against session protection shares much in common with web session hijacking. Session hijacking [116] is an attack of taking over a user session by stealing the session ID and impersonating the authorized user. After that, the attacker gains access to the sensitive information stored in the session. The main countermeasure is improving session management on both server-side and client-side to protect session ID. End-to-end secure channels like SSL can prevent session ID from being intercepted by passive eavesdroppers. The drawback of SSL is that, it brings much additional cost to the communication [108]. As a result, many sites only use SSL to protect the initial login page, and the following communication is conducted over plain HTTP. The session ID may be exposed in the insecure network again.

Sessionlock [63] provides a light-weighted approach to securing web sessions against eavesdropping. After the server sets up the session with the client, it stores the session ID in the client browser as the form of fragment identifier. The browser signs outgoing HTTP requests with the fragment identifier via HMAC to present its identity to the server. Since the browser never sends fragment identifier over the Internet, the eavesdropper cannot intercept the session ID. However, this method is not appropriate for fighting against hotlinking that explores session vulnerability. The hotlinking site can set up a legal session with the hosting site, and embeds the assigned session ID into hotlinking pages for the user to share. In other words, the user can send requests signed by the legal session ID without visiting the hosting site.

Cookie vulnerabilities exploited by hotlinking is in the field of cookie security. HTTP cookies

have serious security and privacy concerns [69]. Cookie theft is the act of intercepting cookies by an unauthorized party. Cookies may be stolen via packet sniffing over the Internet. This scenario is similar to session hijacking. Cross-site scripting attack is another way to steal cookies [140, 102]. Usually the attacker posts malicious code into a webpage, and by running it, the browser itself will send cookies to the attacker. HttpOnly flag is the countermeasure that protects cookies from cross-site scripting, and it makes cookies inaccessible to client-side programs (such as JavaScript). It was first introduced by Microsoft [112] and supported in PHP since version 5.2.0. However, it has not become the industry standard.

Currently, there are some web sites that provide online storage and file delivery service, such as [42, 38]. They have also deployed anti-hotlinking measures. However, their implementation details remains sealed.

## **2.7 Conclusion**

In this chapter we investigate the hotlinking phenomenon with the focus on unauthorized hotlinking. We perform a series of large-scale measurements targeting two types of hotlinked objects, images and software packages. Our measurement results show that hotlinking widely exists over the Internet and is severe in some categories of websites like blogging. We also conduct a detailed postmortem analysis on a real hotlink-victim site, which shows that unexpected large amount of hotlinking traffics can easily overwhelm the victim server. Moreover, we analyze a set of regular hotlinking attacks that explore the weakness of current defense methods. To defend against hotlinking attacks, we present an anti-hotlinking framework based on the existing network security techniques. The framework is highly customizable with different granularities of protection that

webmasters can specify. A prototype of the framework is implemented with the support of the two download policies, Strict Policy and Loose Policy. Its effectiveness against hotlinking attacks is evaluated in terms of security and usability.

## Chapter 3

Twitter is a popular online social networking and micro-blogging tool, which was released in 2006. Remarkable simplicity is its distinctive feature. Its community interacts via publishing text-based posts, known as *tweets*. The tweet size is limited to 140 characters. Hashtag, namely words or phrases prefixed with a # symbol, can group tweets by topic. For example, #Justin Bieber and #Women's World Cup are the two trending hashtags on Twitter in 2011 [51]. Symbol @ followed by a username in a tweet enables the direct delivery of the tweet to that user. Unlike most online social networking sites (i.e., Facebook and MySpace), Twitter's user relationship is directed and consists of two ends, friend and follower. In the case where the user A adds B as a friend, A is a *follower* of B while B is a *friend* of A. In Twitter terms, A follows B. B can also add A as his friend (namely, following back or returning the follow), but is not required. From the standpoint of information flow, tweets flow from the source (author) to subscribers (followers). More specifically, when a user posts tweets, these tweets are displayed on both the author's homepage and those of his followers.

As reported in August 2011, Twitter has attracted 200 million users and generated 8.3 million Tweets per hour [54]. It ranks the 10th on the top 500 site list according to Alexa in December 2011 [50]. In November 2009, Twitter emphasized its value as a news and information network by changing the question above the tweet input dialog box from "What are you doing" to "What's happening". To some extent, Twitter has transformed from a personal micro-blogging site to an

information publish venue. Many traditional industries have used Twitter as a new media channel. We have witnessed successful Twitter applications in business promotion [6], customer service [17], political campaigning [15], and emergency communication [132, 97].

The growing user population and open nature of Twitter have made itself an ideal target of exploitation from automated programs, known as bots. Like existing bots in other web applications (i.e., Internet chat [89], blogs [129] and online games [88]), bots have been common on Twitter. Twitter does not inspect strictly on automation. It only requires the recognition of a CAPTCHA image during registration. After gaining the login information, a bot can perform most human tasks by calling Twitter APIs. More interestingly, in the middle between humans and bots have emerged cyborgs, which refer to either bot-assisted humans or human-assisted bots. Cyborgs have become common on Twitter. After a human registers an account, he may set automated programs (i.e., RSS feed/blog widgets) to post tweets during his absence. From time to time, he participates to tweet and interact with friends. Cyborgs interweave characteristics of both humans and bots.

Automation is a double-edged sword to Twitter. On one hand, legitimate bots generate a large volume of benign tweets, like news and blog updates. This complies with the Twitter's goal of becoming a news and information network. On the other hand, malicious bots have been greatly exploited by spammers to spread spam or malicious contents. These bots randomly add users as their friends, expecting a few users to follow back<sup>1</sup>. In this way, spam tweets posted by bots display on users' homepages. Enticed by the appealing text content, some users may click on links and get redirected to spam or malicious sites<sup>2</sup>. If human users are surrounded by malicious bots and spam tweets, their twittering experience deteriorates, and eventually the whole Twitter community will be

---

<sup>1</sup>Some advanced bots target potential users by keyword search.

<sup>2</sup>Due to the tweet size limit, it is very common to use link shortening service on Twitter, which converts an original link to a short one (i.e., <http://bit.ly/dtUm5Q>). The link illegibility favors bots to allure users.



hurt. The objective of this chapter is to characterize the automation feature of Twitter accounts, and to classify them into three categories, human, bot, and cyborg, accordingly. This will help Twitter manage the community better and help human users recognize who they are tweeting with.

In the chapter, we first conduct a series of measurements to characterize the differences among human, bot, and cyborg in terms of tweeting behavior, tweet content, and account properties. By crawling Twitter, we collect over 500,000 users and more than 40 million tweets posted by them. Then we perform a detailed data analysis, and find a set of useful features to classify users into the three classes. Based on the measurement results, we propose an automated classification system that consists of four major components: (1) the entropy component uses tweeting interval as a measure of behavior complexity, and detects the periodic and regular timing that is an indicator of automation; (2) the spam detection component uses tweet content to check whether text patterns contain spam or not<sup>3</sup>; (3) the account properties component employs useful account properties, such as tweeting device makeup, URL ration, to detect deviations from normal; (4) the decision maker is based on Random Forest, and it uses the combination of the features generated by the above three components to categorize an unknown user as human, bot or cyborg. We validate the efficacy of the classification system through our test dataset. We further apply the system to classify the entire dataset of over 500,000 users collected, and speculate the current composition of Twitter user population based on our classification results.

The remainder of this chapter is organized as follows. Section 3.1 covers related work on Twitter and online social networks. Section 3.2 details our measurements on Twitter. Section 3.3 describes our automatic classification system on Twitter. Section 3.4 presents our experimental results on

---

<sup>3</sup>Spam is a good indicator of automation. Most spam messages are generated by bots, and very few are manually posted by humans.

classification of humans, bots, and cyborgs on Twitter. Finally, Section 3.5 concludes the chapter.

### **3.1 Related Work**

Twitter has been widely used since 2006, and there are some related literature in twittering [101, 106, 147]. To better understand micro-blogging usage and communities, Java et al. [101] studied over 70,000 Twitter users and categorized their posts into four main groups: daily chatter (e.g., “going out for dinner”), conversations, sharing information or URLs, and reporting news. Their work also studied (1) the growth of Twitter, showing a linear growth rate; (2) its network properties, showing the evidence that the network is scale-free like other social networks [114]; and (3) the geographical distribution of its users, showing that most Twitter users are from the US, Europe, and Japan. Krishnamurthy et al. [106] studied a group of over 100,000 Twitter users and classified their roles by follower-to-following ratios into three groups: (1) broadcasters, which have a large number of followers; (2) acquaintances, which have about the same number on either followers or following; and (3) miscreants and evangelists (e.g., spammers), which follow a large number of other users but have few followers. Wu et al. [142] studied the information diffusion on Twitter, regarding the production, flow, and consumption of information. Kwak et al. [107] conducted a thorough quantitative study on Twitter by crawling the entire Twittersphere. Their work analyzed the follower-following topology, and found non-power-law follower distribution and low reciprocity, which all mark a deviation from known characteristics of human social networks. Kim et al. [82] analyzed Twitter lists as a potential source for discovering latent characters and interests of users. A Twitter list consists of multiple users and their tweets. Their research indicated that words extracted from each list are representative of all the members in the list even if the words are not used by the

members. It is useful for targeting users with specific interests.

In addition to network-related studies, several previous works focus on socio-technological aspects of Twitter [150, 132, 97, 127, 100], such as its use in the workplace or during major disaster events.

Twitter has attracted spammers to post spam content, due to its popularity and openness. Fighting against spam on Twitter has been investigated in recent works [147, 92, 136]. Yardi et al. [147] detected spam on Twitter. According to their observations, spammers send more messages than legitimate users, and are more likely to follow other spammers than legitimate users. Thus, a high follower-to-following ratio is a sign of spamming behavior. Grier et al. [92] investigated spam on Twitter from the perspective of spam and click-through behaviors, and evaluated the effectiveness of using blacklists to prevent spam propagation. Their work found out that around 0.13% of spam tweets generate a visit, orders of magnitude higher than click-through rate of 0.003% - 0.006% reported for spam email. Exploiting the social trust among users, social spam may achieve a much higher success rate than traditional spam methods. Thomas et. al [136] studied the behaviors of spammers on Twitter by analyzing the tweets originated from suspended users in retrospect. They found that the current marketplace for Twitter spam uses a diverse set of spamming techniques, including a variety of strategies for creating Twitter accounts, generating spam URLs, and distributing spam.

Compared to previous measurement studies on Twitter, our work covers a relatively large group of Twitter users (more than 500,000) and differs in how we link the measurements to automation, i.e., whether posts are from humans, bots, or cyborgs. While some similar metrics are used in our work, such as follower-to-following ratio, we also introduce some metrics, including entropy of tweet intervals, which are not employed in previous research. Our work also detects spam content

through Bayesian classification. However, our work focuses on determining the automation degree of Twitter accounts, and uses spam as one of the features in the classification.

Twitter is a social networking service, so our work is also related to recent studies on social networks, such as Flickr, LiveJournal, Facebook, MySpace, and YouTube [72, 114, 73]. In [114], with over 11 million users of Flickr, YouTube, LiveJournal, and Orkut, Mislove et al. analyzed link structure and uncovered the evidence of power-law, small-world, and scale-free properties. In [73], Cha et al. examined the propagation of information through the social network of Flickr. Their work shows that most pictures are propagated through the social links (i.e., links received from friends rather than through searches or external links to Flickr content) and the propagation is very slow at each hop. As a result of this slow propagation, a picture's popularity is often localized in one network and grows slowly over a period of months or even years. In [72], Cha et al. analyzed video popularity life-cycles, content aliasing, and the amount of illegal content on YouTube, a popular video sharing service. While YouTube is designed to share large content, i.e., videos, Twitter is designed to share small content, i.e., text messages. Unlike other social networking services, like Facebook or YouTube, Twitter is a micro-content social network, with messages being limited to 140 characters.

As Twitter is a text-based message system, it is natural to compare it with other text-based message systems, such as instant messaging or chat services. Twitter has similar message length (140 characters) to instant messaging and chat services. However, Twitter lacks "presence" (users show up as online/offline for instant messaging services or in specific rooms for chat) but offers (1) more access methods (web, SMS, and various APIs) for reading or posting and (2) more persistent content. Similar to Twitter, instant messaging and chat services also have problems with bots and spam [89, 143]. To detect bots in online chat, Gianvecchio et al. [89] analyzed humans and bots in

Yahoo! chat and developed a classification system to detect bots using entropy-based and machine-learning-based classifiers, both of which are used in our classification system as well. In addition, as Twitter is text-based, email spam filtering techniques are also relevant [91, 149, 144]. However, Twitter posts are much shorter than emails and spaced out over longer periods of time than for instant messages, e.g., hours rather than minutes or seconds.

Twitter also differs from most other network services in that automation, e.g., message feeds, is a major feature of legitimate Twitter usage, blurring the lines between bot and human. Twitter users can be grouped into four categories: humans, bots, bot-assisted humans, and human-assisted bots. The latter two, bot-assisted humans and human-assisted bots, can be described as cyborgs, a mix between bots and humans [145].

## **3.2 Measurement**

In this section, we first describe the data collection of over 500,000 Twitter users. Then, we detail our observation of user behaviors and account properties, which are pivotal to automatic classification.

### **3.2.1 Data Collection**

Here we present the methodology used to crawl the Twitter network and collect detailed user information. Twitter has released a set of API functions [138] that support user information collection. Thanks to Twitter's courtesy of including our test account to its white list, we can make API calls up to 20,000 per hour. This eases our data collection. To diversify our data sampling, we employ two methods to collect the dataset covering more than 500,000 users. The first method is Depth-First

Search (DFS) based crawling. The reason we choose DFS is that it is a fast and uniformed algorithm for traversing a network. Besides, DFS traversal implicitly includes the information about network locality and clustering. Inspired by [90, 94], we randomly select five users as seeds. For each reached user, we record its follower list. Taking the following direction, the crawler continues with the depth constraint set as three. We customize our crawler with a core module of PHP cURL. Ten crawler processes work simultaneously for each seed. After a seed is finished, they move to the next. The crawl duration lasts one month, and 429,423 users are logged.

Similar to the work in [106] and [147], we also use the public timeline API to collect the information of active users, increasing the diversity of the user pool. Twitter constantly posts the twenty most recent tweets in the global scope. The crawler calls the timeline API to collect the authors of the tweets included in the timeline. Since the Twitter timeline frequently updates, the crawler can repeatedly call the timeline API. During the same time window of the DFS crawl, this method contributes 82,984 users to the dataset. We totally collect 512,407 users on Twitter combining both methods.

### 3.2.2 Ground Truth Creation

To develop an automatic classification system, we need a ground truth set that contains known samples of human, bot, and cyborg. Among collected data, we randomly choose different samples and classify them by manually checking their user logs and homepages. The ground truth set includes two thousand users per class of human, bot and cyborg, and thus in total there are six thousand classified samples. In summary, the dataset contains 8,350,095 tweets posted by the sampled users in their account life-time<sup>4</sup>, from which we can extract useful features for classification, such as

---

<sup>4</sup>4,431,923 tweets in the training set, and 3,918,172 tweets in the test set.

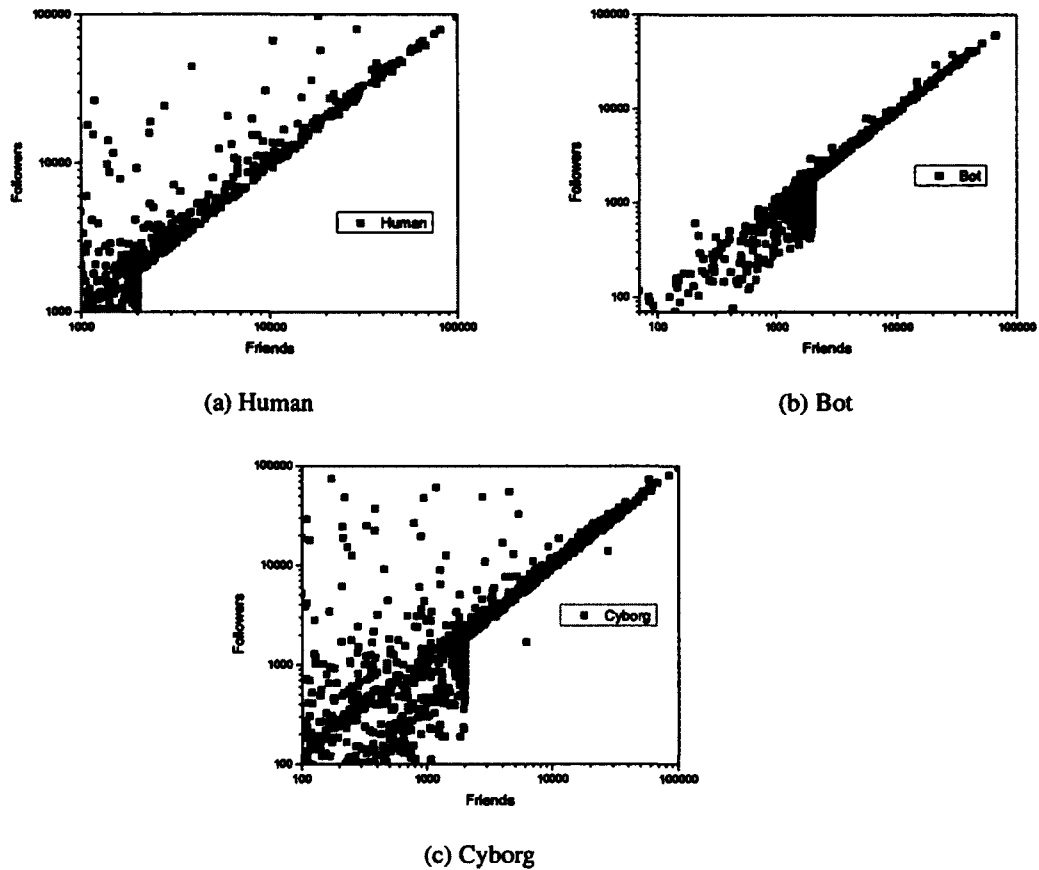


Figure 3.1: Numbers of Followers and Friends

tweeting behaviors and text patterns.

Our log-based classification follows the principle of the Turing test [137]. The standard Turing tester communicates with an unknown subject for five minutes, and decides whether it is a human or machine. Classifying Twitter users is actually more challenging than it appears to be. For many users, their tweets are less likely to form a relatively consistent context. For example, a series of successive tweets may be hardly relevant. The first tweet is the user status, like “watching a football game with my buds.” The second tweet is an automatic update from his blog. The third tweet is a news report RSS feed in the format of article title followed by a shortened URL.

For every account, the following classification procedure is executed. We thoroughly observe the log, and visit the user's homepage (<http://twitter.com/#!/username>) if necessary. We carefully check tweet contents, visit URLs included in tweets (if any), and decide if redirected web pages are related with their original tweets and if they contain spam or malicious contents. We also check other properties, like tweeting devices, user profile, and the numbers of followers and friends. Given a long sequence of tweets (usually we check 60 or more if needed), the user is labeled as a human if we can obtain some evidence of original, intelligent, specific and human-like contents. In particular, a human user usually records what he is doing or how he feels about something on Twitter, as he uses Twitter as a micro-blogging tool to display himself and interact with friends. For example, he may write a post like "I just saw Yankees lost again today. I think they have to replace the starting pitcher for tomorrow's game." The content carries intelligence and originality. Specificity means that the tweet content is expressed in relatively unambiguous words with the presence of consciousness [137]. For instance, in reply to a tweet like "How you like iPad?", a specific response made by human may be "I like its large touch screen and embedded 3G network". On the other hand, a generic reply could be "I like it".

The criteria for identifying a bot are listed as follows. The first is the lack of intelligent or original content. For example, completely retweeting tweets of others or posting adages indicates a lack of originality. The second is the excessive automation of tweeting, like automatic updates of blog entries or RSS feeds. The third is the abundant presence of spam or malicious URLs (i.e., phishing or malware) in tweets or the user profile. The fourth is repeatedly posting duplicate tweets. The fifth is posting links with unrelated tweets. For example, the topic of the redirected web page does not match the tweet description. The last is the aggressive following behavior. In order to gain attention from human users, bots do mass following and un-following within a short period of time. Cy-



borgs are either human-assisted bots or bot-assisted humans. The criterion for classifying a cyborg is the evidence of both human and bot participation. For example, a typical cyborg account may contain very different types of tweets. A large proportion of tweets carry contents of human-like intelligence and originality, while the rest are automatic updates of RSS feeds. It represents a usage model, in which the human uses his account from time to time while the Twitter widget constantly runs on his desktop and posts RSS feeds of his favorite news channel. Lastly, the uncertain category is for non-English users and those without enough tweets to classify. The samples that are difficult and uncertain to classify fall into this category, and are discarded. Some Twitter accounts are set as “private” for privacy protection, and their web pages are only visible to their friends. We do not include such type of users in the classification either, because of their inaccessibility.

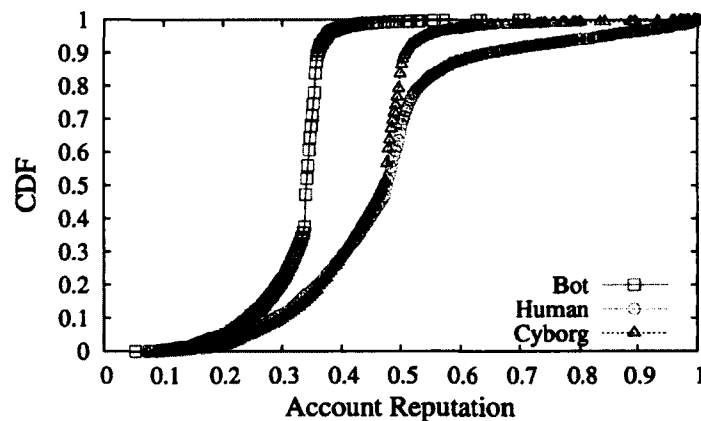


Figure 3.2: CDF of Account Reputation

### 3.2.3 Data Analysis

As mentioned before, Twitter API functions support detailed user information query, ranging from profile, follower and friend lists to posted tweets. In the above crawl, for each user visited, we call API functions to collect abundant information related with user classification. Most information

is returned in the format of XML or JSON. We develop some toolkits to extract useful information from the above well-organized data structures. Our measurement results are presented in the question-answer format.

*Q1. In terms of social relationship, do bots have more friends than followers? A user's tweets can only be delivered to those who follow him. A common strategy shared by bots is following a large number of users (either targeted with purpose or randomly chosen), and expecting some of them will follow back. Figure 3.1 shows the scatter plots of the numbers of followers and friends for the three categories. For better illustration, the scale is chopped and a small amount of extraordinary points are not included. Figure 3.1 contains three different groups of users: group I where the number of one's followers is clearly greater than the number of its friends; group II where the situation is reverse; and group III where the nodes stick around the diagonal.*

In the human category, as shown in Figure 3.1(a), the majority of the nodes belong to group III, implying that the number of their followers is close to that of their friends. This result complies with [114], revealing that human relationships are typically reciprocal in social networks. Meanwhile, there are quite a few nodes belonging to group I with far more followers than friends. They are usually accounts of celebrities and famous organizations. They generate interesting media contents and attract numerous subscribers. For example, the singer Justin Timberlake has 1,645,675 followers and 39 friends (the ratio is 42,197-to-1).

In the bot category, many nodes belong to group II, as shown in Figure 3.1(b). Bots add many users as friends, but few follow them back. Unsolicited tweets make bots unpopular among the human world. However, for some bots, the number of their followers is close to that of their friends. This is due to the following reason. Twitter imposes a limit on the ratio of followers over friends to suppress bots. Thus, some more advanced bots unfollow their friends if they do not follow back

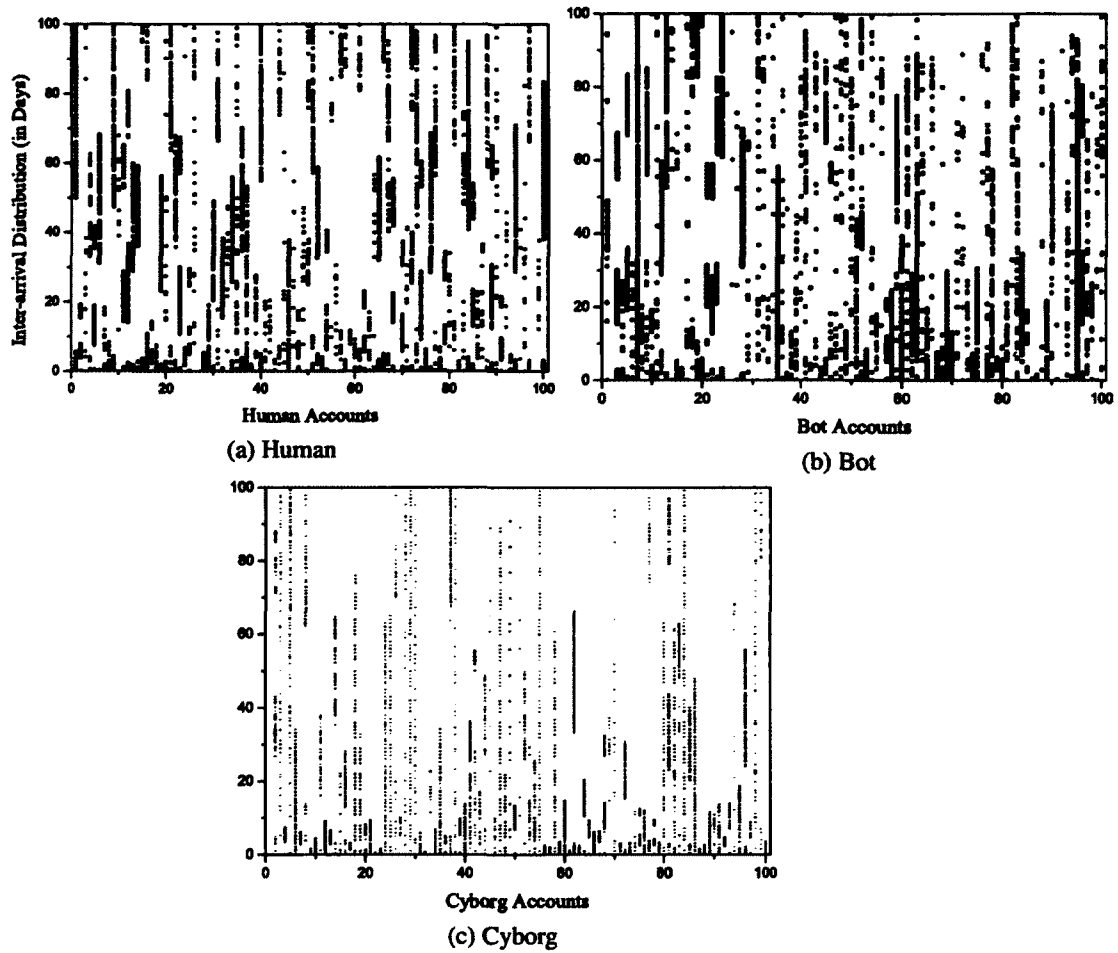


Figure 3.3: Inter-arrival Timing Distribution of Accounts

within a certain amount of time. Those bots cunningly keep the ratio close to 1.

Besides, we have observed that, normal human users are more likely to follow “famous” or “reputable” accounts. We define and normalize

$$Account\ Reputation = \frac{follower\_no}{follower\_no + frined\_no}. \tag{3.1}$$

A celebrity usually has many followers and few friends (such as Justin Timberlake), and his reputation is close to 1. In contrast, for a bot with few followers and many friends, its reputation is

close to 0. Figure 3.2 shows the cumulative distribution function (CDF) of account reputation for three categories. The human category has the largest account reputation, closely followed by cyborg. However, bot's value is much lower. Around 60% of bots have fewer followers than friends, causing account reputation less than 0.5.

*Q2. Does automation generate more tweets?* To answer this question, we measure the number of tweets posted in a user's lifetime<sup>5</sup>. Figure 3.4 shows the CDF of the tweet counts, corresponding to the human, bot and cyborg category. It is clear that cyborg posts more tweets than human and bot. A large proportion of cyborg accounts are registered by commercial companies and websites as a new type of media channel and customer service. Most tweets are posted by automated tools (i.e., RSS feed widgets, Web 2.0 integrators), and the volume of such tweets is considerable. Meanwhile, those accounts are usually maintained by some employees who communicate with customers from time to time. Thus, the high tweet count in the cyborg category is attributed to the combination of both automatic and human behaviors in a cyborg. It is surprising that bot generates fewer tweets than human. We check the bot accounts, and find out the following fact. In its active period, bot tweets more frequently than human. However, bots tend to take long-term hibernation. Some are either suspended by Twitter due to extreme or aggressive activities, while the others are in incubation and can be activated to form bot legions.

*Q3. Does automation generate higher tweeting frequency?* Extended from the previous question, here we examine account's tweeting frequency in active status. Figure 3.3 plots the inter-arrival timing distribution of three categories. Due to space limit, each category contains 100 accounts. Tweets posted by an account are sorted on timestamp, and the timestamp of the first tweet is set as 0. An account is denoted as a vertical strip in the figure, and each of its tweets is denoted as a

---

<sup>5</sup>It is the duration from the time when his account was created to the time when our crawler visited it.

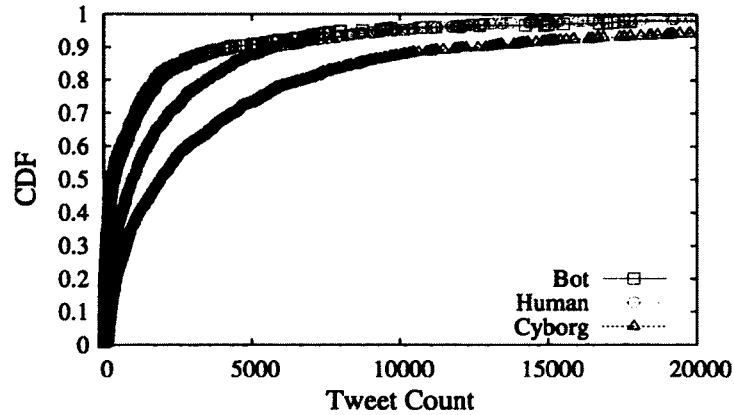


Figure 3.4: CDF of Tweet Count

tiny segment in the strip. We observe the wide existence of burstiness (namely, a block of intensive tweets) in bot, whereas human exhibits less intensive inter-arrival distribution. Automated programs used by bot accounts can constantly operate in the background and intensively post tweets. Most human users tweet with large inter-arrivals (such as hours), and manual behavior cannot generate tweeting frequency as high as bot.

*Q4. Is tweeting behavior regular or complex?* In our measurement we have observed that, many bots are driven by timers to post tweets at fixed inter-arrivals, and thus exhibit regular behavior. In contrast, human behavior carries the inherent complexity [88, 89, 123]. We use entropy rate to measure periodic or regular timing of account's posting behavior. More theoretical details of entropy are presented in Section 3.3.1. For normalization, we define relative entropy as the entropy rate of an account over the maximum entropy rate in the ground truth set. Figure 3.5 demonstrates the CDF of relative entropy of the three categories. Entropy clearly separates bot from human. High entropy indicates irregularity, a sign of manual behavior, whereas low entropy indicates regularity, a sign of automation.

*Q5. How do users post tweets? manually or via auto piloted tools?* Twitter supports a variety

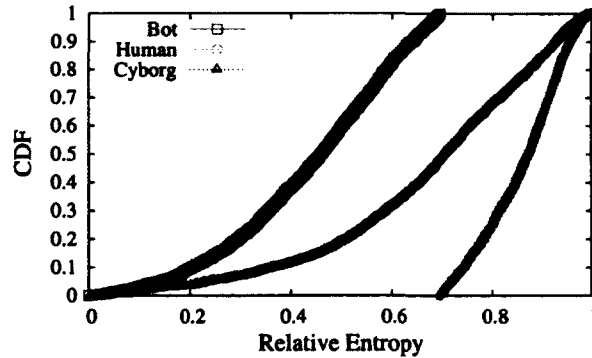
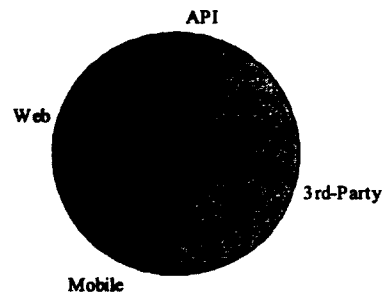


Figure 3.5: CDF of Account's Relative Entropy

of channels to post tweets. The device name appears below a tweet prefixed by “from.” Our whole dataset includes 41,991,545 tweets posted by 3,648 distinct devices. The devices can be roughly divided into the following four categories. (1) Web, a user logs into Twitter and posts tweets via the website. (2) Mobile devices, there are some programs exclusively running on mobile devices to post tweets, like Txt for text messages, Mobile web for web browsers on handheld devices, TwitterBerry for BlackBerry, and twidroid for Android mobile OS. (3) Registered third-party applications, many third-parties have developed their own applications using Twitter APIs to tweet, and registered them with Twitter. From the application standpoint, we can further categorize this group into sub groups including website integrators (twitpic, bit.ly, Facebook), browser extensions (Tweetbar and Twitterfox for Firefox), desktop clients (TweetDeck and Seesmic Desktop), and RSS feeds/blog widgets (twitterfeed and Twitter for Wordpress). (4) APIs, for those third-party applications not registered or certificated by Twitter, they appear as “API” in Twitter.

Figure 3.6 shows the makeup of the above tweeting device categories. Among them, the website of Twitter is the most widely used and generates nearly half of the tweets (46.78%), followed by third-party devices (40.18%). Mobile devices and unregistered API tools contribute 6.81% and 6.23%, respectively. Table 3.1 lists the top ten devices used by the human, bot, and cyborg cate-



**Figure 3.6:** Tweeting Device Makeup

gories, and the whole dataset<sup>6</sup>.

More than half of the human tweets are manually posted via the Twitter website. The rest of top devices are mobile applications (Tweetie, UberTwitter, Mobile web, Txt, TwitterBerry) and desktop clients (TweetDeck, Echofon and Seesmic). In general, tweeting via such devices requires human participation. In contrast, the top tools used by bots are mainly auto piloted, and 42.39% of bot tweets are generated via unregistered API-based tools. Bots can abuse APIs to do almost everything they want on Twitter, like targeting users with keywords, following users, unfollowing those who do not follow back, or posting prepared tweets. Twitterfeed, RSS2Twitter, and Proxifeed are RSS feed widgets that automatically pipeline information (usually in the format of the page title followed by the URL) to Twitter via RSS feeds. Twitter Tools and Twitrne for WordPress are popular WordPress plug-ins that integrate blog updates to Twitter. Assetize is an advertising syndicator mainly targeting at Twitter, and twitRobot is a bot tool that automatically follows other users and posts tweets. All these tools only require minimum human participation (like importing Twitter account information, or setting RSS feeds and update frequency), and thus indicate great automation.

Overall, humans tend to tweet manually and bots are more likely to use auto piloted tools.

---

<sup>6</sup>The whole dataset contains around 500,000 users, and the human, bot and cyborg categories equally contain 1,000 users in the training dataset.

Table 3.1: Top 10 Tweeting Devices

Rank	Human	Bot	Cyborg	All
#1	Web(50.5%)	API(42.4%)	Twitterfeed(31.3%)	Web(46.8%)
#2	TweetDeck(9.2%)	Twitterfeed(26.1%)	Web(23.0%)	TweetDeck(9.3%)
#3	Tweetie(6.2%)	twitRobot(13.1%)	API(6.9%)	Twitterfeed(7.8%)
#4	UberTwitter(3.6%)	RSS2Twitter(2.7%)	Assetize(5.7%)	API(6.2%)
#5	Mobile web(3.0%)	Twitter Tools(1.2%)	HootSuite(5.2%)	Echofon(2.8%)
#6	Txt(2.6%)	Assetize(1.2%)	WP to Twitter(2.4%)	Tweetie(2.5%)
#7	Echofon(2.2%)	Proxifeed(1.1%)	TweetDeck(1.5%)	Txt(2.1%)
#8	TwitterBerry(2.1%)	TweetDeck(1.0%)	UberTwitter(1.2%)	HootSuite(2.1%)
#9	Twitterrific(1.9%)	bit.ly(0.9%)	RSS2Twitter(1.2%)	UberTwitter(1.7%)
#10	Seesmic(1.6%)	Twitme for WordPress(0.8%)	Twaitter(0.9%)	Mobile web(1.5%)

Cyborgs employ the typical human and bot tools. The cyborg group includes many human users who access their Twitter accounts from time to time. For most of the time when they are absent, they leave their accounts to auto piloted tools for management.

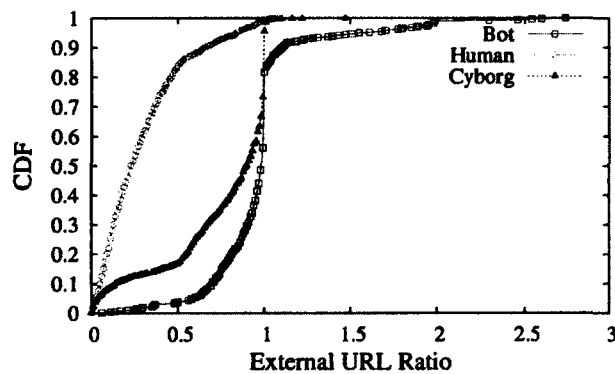


Figure 3.7: External URL ratio in tweets

*Q6. Do bots include more external URLs than humans?* In our measurement, we find out that, most bots tend to include URLs in tweets to redirect visitors to external web pages. For example, spam bots are created to spread unsolicited commercial information. Their topics are similar to those in email spam, including online marketing and affiliate programs, working at home, selling fake luxury brands or pharmaceutical products<sup>7</sup>. However, the tweet size is up to 140 characters,

<sup>7</sup>A new topic is attracting more followers on Twitter. It follows the style of pyramid sales by asking newly joined users to follow existing users in the spam network.



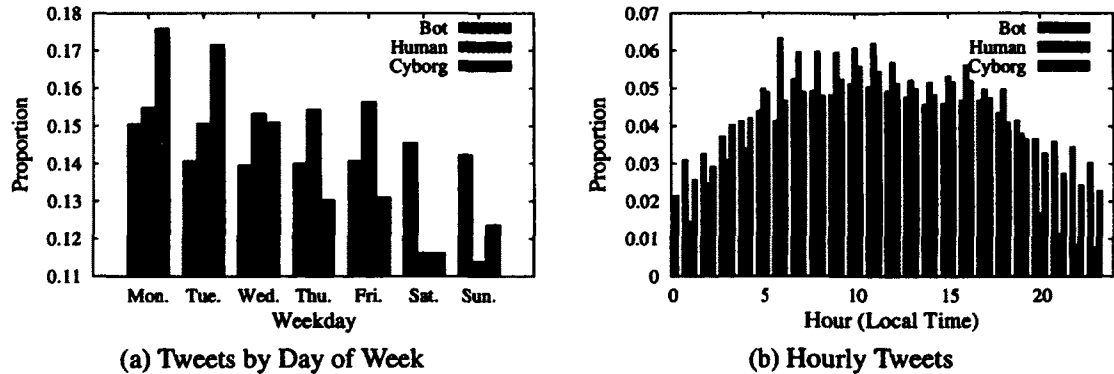


Figure 3.8: Tweets Posted on Daily/Hourly Base

which is rather limited for spammers to express enough text information to allure users. Basically, a spam tweet contains an appealing title followed by an external URL. Figure 3.7 shows the external URL ratios (namely, the number of external URLs included in tweets over the number of tweets posted by an account) for the three categories, among which the URL ratio of bot is highest. Some tweets by bots even have more than one URL<sup>8</sup>. The URL ratio of cyborg is very close to the bot's level. A large number of cyborgs integrate RSS feeds and blog updates, which take the style of webpage titles followed by page links. The URL ratio of human is much lower, on average it is only 29%. When a human tweets what is he doing or what is happening around him, he mainly uses text and does not often link to web pages.

*Q7. Are there any other temporal properties of Twitter users helpful for differentiation among human, bot, and cyborg?* Many research works like [84] and [81] have shown the weekly and diurnal access patterns of humans in the Internet. Figures 3.8(a) and 3.8(b) present the tweeting percentages of the three different categories on daily and hourly bases, respectively. The weekly behavior of Twitter users shows clear differences among the three categories. While humans are more active during the regular workdays, from Monday to Friday, and less active during the weekend, Sat-

<sup>8</sup>Many such accounts belong to a type of bot that always appends a spam link to tweets it re-tweets.

urday and Sunday, bots have roughly the same activity level every day of the week. Interestingly, cyborgs are the most active ones on Monday and then slowly decrease their tweeting activities during the week; on Saturday cyborgs reach their lowest active point but somehow bounce back a bit on Sunday. Such a cyborg activity trend is mainly caused by their message feeds and the high level of news and blog activities at the start of a week. Similarly, the hourly behavior of human is more active during the daytime, which mostly overlaps with office hours. The bot activity is nearly even except a little drop in the deep of night. Some more advanced bots have the setting of “only tweet from a time point to another,” which helps save API calls [52]. Thus, they can tweet more in the daytime to better draw the attention of humans.

**Figure 3.9:** Account Registration Date (Grouped by Quarter)

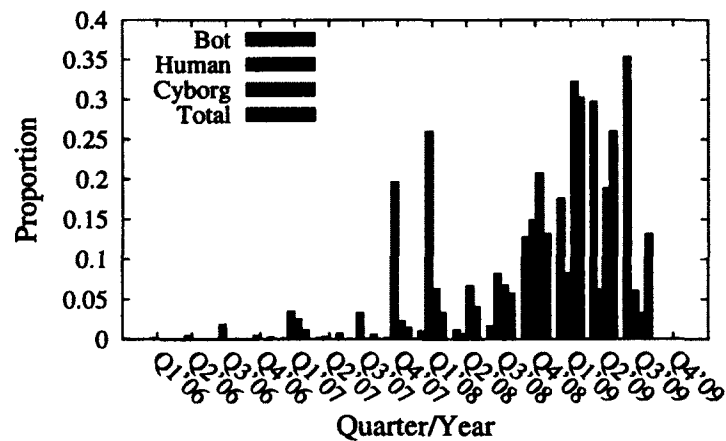


Figure 3.9 shows account registration dates grouped by quarter. We have two observations from the figure. First, the majority of accounts (80.0% of humans, 94.8% of bots, and 71.1% of cyborgs) were registered in 2009. It confirms the dramatic growth of Twitter in 2009. Second, we do not find any bot or cyborg in our ground truth dataset earlier than March, 2007. However, human registration has continued increasing since Twitter was founded in 2006. Thus, old accounts are less likely to be bots.

*Q8. Are users aware of privacy and identity protection on Twitter?* Twitter provides a protected option to protect user privacy. If it is set as true, the user's homepage is only visible to his friends. However, the option is set as false by default. In our dataset of over 500,000 users, only 4.9% of them are protected users. Twitter also verifies some accounts to authenticate users' real identities. More and more celebrities and famous organizations have applied for verified accounts. For example, Bill Gates has his verified Twitter account at <http://twitter.com/billgates>. However, in our dataset, only 1.8% of users have verified accounts.

### 3.3 Classification

This section describes our automated system for classification of Twitter users. The system classifies Twitter users into three categories: human, bot, and cyborg. The system consists of several components: the entropy component, the spam detection component, the account properties component, and the decision maker. The high-level design of our Twitter user classification system is shown in Figure 3.10.

The entropy component uses corrected conditional entropy to detect periodic or regular timing, which is a sign of automation. The spam detection component uses a variant of Bayesian classification to detect text patterns of known spam on Twitter. The account properties component uses account-related properties to catch bot deviation from the normal human distribution. Lastly, the decision maker based on Random Forest algorithm analyzes the features identified by the other three components and makes a decision: human, cyborg, or bot.

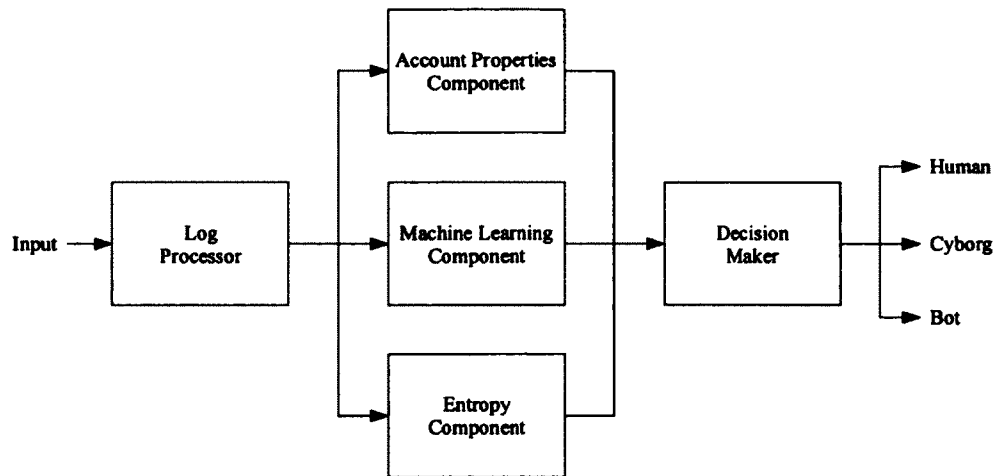


Figure 3.10: Classification System

### 3.3.1 Entropy Component

The entropy component detects periodic or regular timing of the messages posted by a Twitter user. On one hand, if the entropy or corrected conditional entropy is low for the inter-tweet delays, it indicates periodic or regular behavior, a sign of automation. More specifically, some of the messages are posted via automation, i.e., the user may be a potential bot or cyborg. On the other hand, a high entropy indicates irregularity, a sign of human participation.

#### 3.3.1.1 Entropy Measures

The entropy rate is a measure of the complexity of a process [79]. The behavior of bots is often less complex than that of humans [87, 98], which can be measured by entropy rate. A low entropy rate indicates a regular process, whereas a high entropy rate indicates a random process. A medium entropy rate indicates a complex process, i.e., a mix of order and disorder [96].

The entropy rate is defined as either the average entropy per random variable for an infinite sequence or as the conditional entropy of an infinite sequence. Thus, as real datasets are finite, the

conditional entropy of finite sequences is often used to estimate the entropy rate. To estimate the entropy rate, we use the corrected conditional entropy [122]. The corrected conditional entropy is defined as follows.

A random process  $X = \{X_i\}$  is defined as a sequence of random variables. The entropy of such a sequence of random variables is defined as:

$$H(X_1, \dots, X_m) = - \sum_{i=1}^m P(x_i) \log P(x_i), \quad (3.2)$$

where  $P(x_i)$  is the probability  $P(X_i = x_i)$ .

The conditional entropy of a random variable given a previous sequence of random variables is:

$$H(X_m | X_1, \dots, X_{m-1}) = H(X_1, \dots, X_m) - H(X_1, \dots, X_{m-1}). \quad (3.3)$$

Then, based on the conditional entropy, the entropy rate of a random process is defined as:

$$\bar{H}(X) = \lim_{m \rightarrow \infty} H(X_m | X_1, \dots, X_{m-1}). \quad (3.4)$$

The corrected conditional entropy is computed as a modification of Equation 5.3. First, the joint probabilities,  $P(X_1 = x_1, \dots, X_m = x_m)$  are replaced with empirically-derived probabilities. The data is binned into  $Q$  bins, i.e., values are converted to bin numbers from 1 to  $Q$ . The empirically-derived probabilities are then determined by the proportions of bin number sequences in the data. The entropy estimate and conditional entropy estimate, based on empirically-derived probabilities, are denoted as  $EN$  and  $CE$  respectively. Second, a corrective term,  $perc(X_m) \cdot EN(X_1)$ , is added to adjust for the limited number of sequences for increasing values of  $m$  [122]. The corrected

conditional entropy, denoted as  $CCE$ , is computed as:

$$\begin{aligned} CCE(X_m | X_1, \dots, X_{m-1}) = \\ CE(X_m | X_1, \dots, X_{m-1}) + perc(X_m) \cdot EN(X_1), \end{aligned} \quad (3.5)$$

where  $perc(X_m)$  is the percentage of unique sequences of length  $m$  and  $EN(X_1)$  is the entropy with  $m$  fixed at 1 or the first-order entropy.

The estimate of the entropy rate is the minimum of the corrected conditional entropy over different values of  $m$ . The minimum of the corrected conditional entropy is considered to be the best estimate of the entropy rate from the limited number of sequences.

### 3.3.2 Spam Detection Component

The spam detection component examines the content of tweets to detect spam. We have observed that most spam tweets are generated by bots and only very few of them are manually posted by humans. Thus, the presence of spam patterns usually indicates automation. Since tweets are text, determining if their content is spam can be reduced to a text classification problem. The text classification problem is formalized as  $f : T \times C \rightarrow \{0, 1\}$ , where  $f$  is the classifier,  $T = \{t_1, t_2, \dots, t_n\}$  are the texts to be classified, and  $C = \{c_1, c_2, \dots, c_k\}$  are the classes [126]. A value of 1 for  $f(t_i, c_j)$  indicates that text  $t_i$  belongs to class  $c_j$ , whereas a value of 0 indicates it does not belong to that class. Bayesian classifiers are very effective in text classification, especially for email spam detection, so we employ Bayesian classification for our machine learning text classification component.

In Bayesian classification, deciding if a message belongs to a class, e.g., spam, is done by computing the corresponding probability based on its content, e.g.,  $P(C = spam|M)$ , where  $M$  is a message and  $C$  is a class. If the probability is over a certain threshold, then the message is from

that class.

The probability that a message  $M$  is spam,  $P(spam|M)$ , is computed from Bayes theorem:

$$P(spam|M) = \frac{P(M|spam)P(spam)}{P(M)} = \frac{P(M|spam)P(spam)}{P(M|spam)P(bot) + P(M|not spam)P(not spam)}. \quad (3.6)$$

The message  $M$  is represented as a feature vector  $\langle f_1, f_2, \dots, f_n \rangle$ , where each feature  $f$  is one or more words in the message and each feature is assumed to be conditionally independent.

$$P(spam|M) = \frac{P(spam) \prod_{i=1}^n P(f_i|spam)}{P(spam) \prod_{i=1}^n P(f_i|spam) + P(not spam) \prod_{i=1}^n P(f_i|not spam)}. \quad (3.7)$$

The calculation of  $P(spam|M)$  varies in different implementations of Bayesian classification. The implementation used for our machine learning component is CRM114 [148]. CRM114 is a powerful text classification system that offers a variety of different classifiers. The default classifier for CRM114 is Orthogonal Sparse Bigram (OSB), a variant of Bayesian classification, which has been shown to perform well for email spam filtering. OSB differs from other Bayesian classifiers in that it treats pairs of words as features.

### 3.3.3 Account Properties Component

Besides inter-tweet delay and tweet content, some Twitter account-related properties are very helpful for the user classification. As shown in Section 3.2.3, obvious difference exists between the human and bot categories. The first property is the URL ratio. The ratio indicates how often a user includes external URLs in its posted tweets. External URLs appear very often in tweets posted by

a bot. Our measure shows, on average the ratio of bot is 97%, while that of human is much lower at 29%. Thus, a high ratio (e.g., close to one) suggests a bot and a low ratio implies a human. The second property is tweeting device makeup. According to Table 3.1, about 70% tweets of human are posted via web and mobile devices (referred as manual devices), whereas about 87% tweets of bot are posted via API and other auto-piloted programs (referred as auto devices). The third property is the followers to friends ratio.

The fourth property is link safety, i.e., to decide whether external links in tweets are malicious/phishing URLs or not. We run a batch script to check a URL in five blacklists: Google Safe Browsing, PhishingTank, URIBL, SURBL and Spamhaus [31, 39, 59, 48, 45]. Google Safe Browsing checks URLs against Google's constantly updated lists of suspected phishing and malware pages. PhishingTank focuses on phishing websites. The mechanisms of URIBL, SURBL and Spamhaus are similar. They contain those suspicious websites that have appeared in spam emails, primarily Unsolicited Bulk/Commercial Email (UBE/UCE). If the URL appears in any of the blacklists, the feature of link safety is set as false.

The fifth property is whether a Twitter account is verified. No bot in our ground truth dataset is verified. The account verification suggests a human. The sixth property is the account registration date. According to Figure 3.9, 94.8% of bots were registered in 2009. The last two properties are the hashtag ratio and mention ratio. Hashtag ratio of an account is defined as the number of hashtags included in the tweets over the number of tweets posted by the account. Mention ratio is defined similarly.

The account properties component extracts these properties from the user log, and sends them to the decision maker. It assists the entropy component and the spam detection component to improve the classification accuracy.



### 3.3.4 Decision Maker

Our classification problem can be formulated as follows. Given an unknown user  $U$  represented by the feature vector, the decision maker determines the class  $C$  to which  $U$  belongs to. Namely,  $U = \langle f_1, f_2, \dots, f_n \rangle \rightarrow C = \{human, bot, cyborg\}$ . We select Random Forest [70] as the machine learning algorithm, and implement the decision maker based on it.

Random Forest creates an ensemble classifier consisting of a set of decision trees. The algorithm applies the random feature selection in [70] and bagging idea in [95] to construct a “collective forest” of decision trees with controlled variation. The decision tree contains two types of nodes, the leaf node labeled as a class, and the interior node associated with a feature. We denote the number of features in the dataset as  $M$ , and the number of features used to make the decision at a node of the tree as  $m (<< M)$ . Each decision tree is built top-down in a recursive manner. For every node in the construction path,  $m$  features are randomly selected to reach a decision at the node. The node is then associated with the feature that is the most informative. Entropy is used to calculate the information gain contributed by each of the  $m$  features (namely, how informative a feature is). In other words, the recursive algorithm applies a greedy search by selecting the candidate feature that maximizes the heuristic splitting criterion.

We denote  $D$  as the dataset of labeled samples, and  $C$  as the class with  $k$  values,  $C = \{C_1, C_2, \dots, C_k\}$ . The information required to identify the class of a sample in  $D$  is denoted as  $Info(D) = Entropy(P)$ , where  $P$ , as the probability distribution of  $C$ , is

$$P = \left\{ \frac{|C_1|}{|D|}, \frac{|C_2|}{|D|}, \dots, \frac{|C_k|}{|D|} \right\}.$$

If we partition  $D$  based on the value of a feature  $F$  into subsets  $\{D_1, D_2, \dots, D_n\}$ ,

$$Info(F, D) = \sum_{i=1}^n \frac{|D_i|}{|D|} Info(D_i). \quad (3.8)$$

After the value of feature  $F$  is obtained, the corresponding gain in information due to  $F$  is denoted as

$$Gain(F, D) = Info(D) - Info(F, D), \quad (3.9)$$

As *Gain* favors features that have a large number of values, to compensate for this *Gain Ratio* is defined as

$$GainRatio(F, D) = \frac{Gain(F, D)}{SplitInfo(F, D)} \quad (3.10)$$

where  $SplitInfo(F, D)$  is the information due to the splitting of  $D$  based on the value of attribute  $F$ .

Thus,

$$SplitInfo(A, D) = Entropy\left(\frac{|D_1|}{|D|}, \frac{|D_2|}{|D|}, \dots, \frac{|D_n|}{|D|}\right) \quad (3.11)$$

More details of decision tree learning can be found in [105]. To classify an unknown sample, it is push downwards in the tree, and assigned with the class of the leaf node with which it ends up. Every decision tree determines a classification decision on the sample. Random Forest applies the majority voting of all the individual decisions to reach the final decision.

### 3.4 Evaluation

In this section, we first evaluate the accuracy of our classification system based on the ground truth set. Then, we apply the system to classify the entire dataset of over 500,000 users collected. With the classification results, we further speculate the current composition of Twitter user population. Finally, we discuss the robustness of the proposed classification system against possible evasions.

### 3.4.1 Methodology

As shown in Figure 3.10, the components of the classification system collaborate in the following way. The entropy component calculates the entropy (and corrected conditional entropy) of inter-tweet delays of a Twitter user. The entropy component only processes logs with more than 100 tweets<sup>9</sup>. This limit helps reduce noise in detecting automation. A lower entropy indicates periodic or regular timing of tweeting behavior, a sign of automation, whereas a higher entropy implies irregular behavior, a sign of human participation. The spam detection component determines if the tweet content is either spam or not, based on the text patterns it has learned. The content feature value is set to 1 for spam but 0 for non-spam. The account properties component checks all the properties mentioned in Section 3.3.3, and generates a real-number-type value for each property. Given a Twitter user, the above three components generate a set of features and input them into the decision maker. For each class, namely human, bot and cyborg, the decision maker computes a classification score for the user, and classifies it into the class with the highest score. The training of the classification system and cross validation of its accuracy are detailed as follows.

### 3.4.2 Classification System Training

The spam detection component of the classification system requires training before being used. It is trained on spam and non-spam datasets. The spam dataset consists of spam tweets and spam external URLs, which are detected during the creation of the ground truth set. Some advanced spam bots intentionally inject non-spam tweets (usually in the format of pure text without URLs, such

---

<sup>9</sup>The inter-tweet span could be wild on Twitter. An account may be inactive for months, but suddenly tweets at an intensive frequency for a short-term, and then enters hibernation again. It generates noise to the entropy component. Thus, the entropy component does not process logs with less than 100 tweets. Besides, in practice it is nearly impossible to determine automation based on a very limited number of tweets.

as adages<sup>10</sup>) to confuse human users. Thus, we do not include such vague tweets without external URLs. The non-spam dataset consists of all human tweets and cyborg tweets without external URLs. Most human tweets do not carry spam. Cyborg tweets with links are hard to determine without checking linked web pages. They can be either spam or non-spam. Thus, we do not include this type of tweets in either dataset. Training the component with up-to-date spam text patterns on Twitter helps improve the accuracy. In addition, we create a list of spam words with high frequency on Twitter to help the Bayesian classifier capture spam content.

### 3.4.3 Classification System Accuracy

We use Weka, a machine learning tool [93], to implement the Random Forest based classifier. We apply cross validation with ten folds to train and test the classifier over the ground truth set [111]. The dataset is randomly partitioned into ten complementary subsets with equal size. In each round, one out of ten subsets is retained as the test set to validate the classifier, while the remaining nine subsets are used as the training set to train the classifier. At the beginning of a round, the classifier is reset and re-trained. Thus, each round is an independent classification procedure, and does not affect subsequent ones. The individual results from ten rounds are averaged to generate the final estimation. The advantage of cross validation is that, all samples in the dataset are used for both training and validation, while each sample is validated exactly once. The confusion matrix listed in Table 3.2 demonstrates the classification results.

The “Actual” rows in Table 3.2 denote the actual classes of the users, and the “Classified” columns denote the classes of the users as decided by the classification system. For example, the cell

---

<sup>10</sup>A typical content pattern is listed as follows. Tweet 1, A friend in need is a friend in deed. Tweet 2, Danger is next neighbor to security. Tweet 3, Work home and make \$3k per month. Check out how, <http://tinyurl.com/bF234T>.

**Table 3.2: Confusion Matrix**

		Classified			Total	True Pos
		Human	Cyborg	Bot		
Actual	Human	1972	27	1	2000	98.6%
	Cyborg	65	1833	102	2000	91.7%
	Bot	2	46	1952	2000	97.6%
					Avg	96.0%

in the junction of the “Human” row and column means that 1972 humans are classified (correctly) as humans, whereas the cell of “Human” row and “Cyborg” column indicates that 27 humans are classified (incorrectly) as cyborgs. There is no misclassification between human and bot.

We examine the logs of those users being classified by mistake, and analyze each category as follows.

- For the human category, 1.4% of human users are classified as cyborg by mistake. One reason is that, the overall scores of some human users are lowered by spam content penalty. The tweet size is up to 140 characters. Some patterns and phrases are used by both human and bot, such as “I post my online marketing experience at my blog at <http://bit.ly/xT6klM>. Please ReTweet it.” Another reason is that the tweeting interval distribution of some human users is slightly lower than the entropy means, and they are penalized for that.
- For the bot category, 2.3% of bots are wrongly categorized as cyborg. The main reason is that, most of them escape the spam penalty from the machine learning component. Some spam tweets have very obscure text content, like “you should check it out since it’s really awesome. <http://bit.ly/xT6klM>”. Without checking the spam link, the component cannot determine if the tweet is spam merely based on the text.
- For the cyborg category, 3.3% of cyborgs are mis-classified as human, and 5.1% of them are mis-classified as bot. A cyborg can be either a human-assisted bot or a bot-assisted human.

A strict policy could categorize cyborg as bot, while a loose one may categorize it as human.

- There is negligible mis-classification between human and bot. The classifier clearly separates these two classes.

Overall, our classification system can accurately differentiate human from bot. However, it is much more challenging for a classification system to distinguish cyborg from human or bot. After averaging the true positive rates of the three classes with equal sample size, the overall system accuracy can be viewed as (96.0%).

Among the set of features used in classification, some play a more important role than others. Now we evaluate the discrimination weight of each feature. In every test, we only use one feature to independently cross validate the ground truth set. Table 3.3 presents the results sorted on accuracy. The entropy feature has the highest accuracy at 82.8%. It effectively captures the timing difference between regularity of automated behavior and complexity of manual behavior. Limited by tweet size, bot usually relies on URLs to redirect users to external web sites. This fact makes the URL ratio feature have a relatively high accuracy at 74.9%. Recognizing the tweeting device makeup (manual or automated) and detecting spam content also help the classification. By comparing the collective performance in Table 3.2 and individual performance in Table 3.3, we observe that, no single feature works perfectly well, and the combination of multiple features improve the classification accuracy.

#### **3.4.4 Twitter Composition**

We further use the classification system to automatically classify our whole dataset of over 500,000 users. We can speculate the current composition of Twitter user population based on the classification results. The system classifies 53.2% of the users as human, 36.2% as cyborg, and 10.5% as bot.

**Table 3.3: Feature Weights**

Feature	Accuracy (%)
Entropy	82.8
URL Ratio	74.9
Automated Device %	71.0
Bayesian Spam Detection	69.5
Manual Device %	69.2
Registration Date	62.9
Mention Ratio	56.2
Link Safety	49.3
Hashtag Ratio	47.0
Followers to Friends Ratio	45.3
Account Verification	35.0

Thus, we speculate the population proportion of human, cyborg and bot category roughly as 5:4:1 on Twitter.

### 3.4.5 Resistance to Evasion

Now we discuss the resistance of the classification system to possible evasion attempts made by bots. Bots may deceive certain features, such as the followers to friends ratio as mentioned before. However, our system has two critical features that are very hard for bots to evade. The first feature is tweeting device makeup, which corresponds to the manual/auto device percentage in Table 3.3. Manual device refers to web and mobile devices, while auto device refers to API and other auto-piloted programs (see Section 3.2.3, Q5). Tweeting via web requires a user to login and manually post via the Twitter website in a browser. Posting via HTTP form is considered by Twitter as API. Furthermore, currently it is impractical or expensive to run a bot on a mobile device to frequently tweet. As long as Twitter can correctly identify different tweeting platforms, device makeup is an effective metric for bot detection. The second feature is URL ratio. Considering the limited tweet length that is up to 140 characters, most bots have to include a URL to redirect users to external sites. Thus, a high URL ratio is another effective metric for bot detection. Other features like timing

entropy, bot could mimic human behaviors but at the cost of much reduced tweeting frequency. We will continue to explore new features emerging with the Twitter development for more effective bot detection in the future.

### 3.5 Conclusion

In this chapter, we have studied the problem of automation by bots and cyborgs on Twitter. As a popular web application, Twitter has become a unique platform for information sharing with a large user base. However, its popularity and very open nature have made Twitter a very tempting target for exploitation by automated programs, i.e., bots. The problem of bots on Twitter is further complicated by the key role that automation plays in everyday Twitter usage.

To better understand the role of automation on Twitter, we have measured and characterized the behaviors of humans, bots, and cyborgs on Twitter. By crawling Twitter, we have collected one-month of data with over 500,000 Twitter users with more than 40 million tweets. Based on the data, we have identified features that can differentiate humans, bots, and cyborgs on Twitter. Using entropy measures, we have determined that humans have complex timing behavior, i.e., high entropy, whereas bots and cyborgs are often given away by their regular or periodic timing, i.e., low entropy. In examining the text of tweets, we have observed that a high proportion of bot tweets contain spam content. Lastly, we have discovered that certain account properties, like external URL ratio and tweeting device makeup, are very helpful on detecting automation.

Based on our measurements and characterization, we have designed an automated classification system that consists of four main parts: the entropy component, the spam detection component, the account properties component, and the decision maker. The entropy component checks for periodic



or regular tweet timing patterns; the spam detection component checks for spam content; and the account properties component checks for abnormal values of Twitter-account-related properties. The decision maker summarizes the identified features and decides whether the user is a human, bot, or cyborg. The effectiveness of the classification system is evaluated through the test dataset. Moreover, we have applied the system to classify the entire dataset of over 500,000 users collected, and speculated the current composition of Twitter user population based on the classification results.

## Chapter 4

With the tremendous popularity of online social networks (OSNs), spammers have exploited them for spreading spam messages. Social spamming is more successful than traditional methods such as email spamming by taking advantage of social relationship between users. One important reason is that OSNs help build intrinsic trust relationship between cyber friends even though they may not know each other in reality. This leads to users to feel more confident to read messages or even click links from their cyber friends. Facilitated by this fact, spammers have greatly abused OSNs and posted malicious or spam content, trying to reach more victims.

Detecting spam is the first and very critical step in the battle of fighting spam. The definition of spam in this chapter is malicious, phishing or scam content. Our work chooses Twitter as the battlefield. Currently, Twitter is the most popular micro-blogging site with 200 million users. Users can post short textual messages, called *tweets*, of up to 140 characters. Twitter has witnessed a variety of spam attacks. Conventional spam detection methods on Twitter mainly check individual tweets or accounts for the existence of spam [131, 67]. The tweet-level detection screens individual tweets to check whether they contain spam text content or URLs. As of August 2011, around 8.3 million tweets are generated per hour [54], and they demand near real-time delivery. Thus, the tweet-level detection would consume too much computing resources and can hardly meet time-stringent requirements. The account-level detection checks individual accounts for the evidence of posting spam tweets or aggressive automation behavior. Once an account is classified as a spam account

according to the Twitter rules of spam and abuse [56], the account will be suspended. Suspending spam accounts is an endless cat and mouse game as it is easy for spammers to create new accounts to replace suspended ones.

Our work shifts the perspective from individual detection to collective detection and focuses on detecting spam campaigns. A spam campaign is defined as a collection of multiple accounts controlled and manipulated by a spammer to spread spam on Twitter for a specific purpose (e.g., advertising a spam site or selling counterfeit goods). Detecting spam campaigns is an important complement to conventional spam detection methods. Moreover, our work brings two additional benefits:

- **Efficiency.** Our approach clusters related spam accounts into a campaign and generates a signature for the spammer behind the campaign. Thus, not only our work can detect multiple existing spam accounts at a given time, it can also capture future ones if the spammer maintains the same spamming strategies.
- **Robustness.** There are some spamming methods which cannot be detected at individual level. For example, Twitter defines the behavior of “posting duplicate content over multiple accounts” as spamming. By grouping related accounts, our work can detect such a collective spamming behavior.

We have performed data collection for three months in 2011, and obtained a dataset with 50 million tweets posted by 22 million users. Our work focuses on the analysis of the tweets containing external URLs<sup>1</sup> The resulting dataset includes eight million tweets with URLs. For those tweets,

---

<sup>1</sup>Considering the short length of tweet, most spam tweets must use external URLs to redirect users to spam sites. In this chapter, we assume that tweets without external URLs are non-spam. Detecting spam tweets without URLs can be addressed with existing techniques on detecting textual spam content.

we crawl every URL. If URL redirection is used, we convert the original URL to its final landing URL. Then, we cluster tweets with the same final URL into a campaign, partitioning the dataset into numerous campaigns based on URLs. We perform a detailed analysis over the campaign data and generate a set of useful features to classify a campaign into two classes: spam or legitimate. Based on the measurement results, we present an automatic classification system using machine learning. We validate the efficacy of the classification system. The experimental results show high accuracy with low false positive rate.

The remainder of the chapter is organized as follows. Section 4.1 presents a brief background of Twitter and covers related work of social spam detection. Section 4.2 details the data collection and measurements on Twitter. Section 4.3 describes our automatic classification system. Section 4.4 evaluates the system efficacy for detecting spam campaigns. Finally, Section 4.5 concludes the chapter.

## **4.1 Related Work**

As spammers often use Twitter-specific features to allure victims, we first briefly describe the background of Twitter and its working mechanism. Then, we survey related work in social spam detection and discuss the scope of our work.

### **4.1.1 Background of Twitter**

Twitter, released in 2006, has become a popular micro-blogging tool and online social network. Users post textual messages, known as tweets. The tweet length is up to 140 characters, which limits the spam content the spammer can include in a tweet. Thus, embedding an external URL in

a tweet becomes a routine for spammers to allure users to spam websites. Meanwhile, to facilitate URL posting, Twitter provides URL shortening services for users to convert an arbitrarily long URL to a short one<sup>2</sup>.

A tweet may contain some textual features for better user interaction experience, which are also abused by spammers. A *hashtag*, namely a word or a phrase prefixed with the # symbol, is used to group tweets by their topic. For example, #Japan.Tsunami and #Egyptian.Revolution are two of the worldwide trending hashtags on Twitter in March 2011. Spammers may attach popular hashtags to unrelated spam tweets to increase the chance of being searched. This spamming trick is called hashtag hijacking. The *mention* feature, namely the @ symbol followed by a username in a tweet, enables the direct delivery of the tweet to the user. This feature facilitates spammers to directly send spam to targeted users.

Different from most OSNs (such as Facebook and Myspace), the social relationship on Twitter is directed and need not be bidirectional. It consists of two roles, friend and follower. In the case where Alice adds Bob as a friend (namely, Alice follows Bob in Twitter's term), Alice is a follower of Bob, while Bob is a friend of Alice. Bob may also add Alice as a friend (namely, following back or returning the follow), but is not obligatory. From the perspective of information dissemination, followers of a user subscribe to receive tweets posted by the author. It is the goal of spammers to achieve a large number of followers, which allows broadcasting spam to many users. Acquiring followers is a big challenge for spammers. A common trick shared by spammers is to massively add normal users as friend and expect some of them to follow back.

---

<sup>2</sup>For example, <http://www.newyork.yankees.mlb.com> can be shortened as <http://atmlb.com/7sOhzZ>.

### 4.1.2 Social Spam Detection

Traditional spam methods include sending spam emails [144] and creating spam web content [118]. The past few years have witnessed the rapid rise of online social networks. One key feature of such systems is the reliance on content contributed by users. Unfortunately, the system openness coupled with the large user population has made OSNs an ideal target of social spammers. By exploiting the social trust among users, social spam may achieve a much higher success rate than traditional spam methods. For example, Grier *et al.* analyzed the click-through rate of spam on Twitter [92], and found out that around 0.13% of spam tweets generate a visit, orders of magnitude higher than click-through rate of 0.003% - 0.006% reported for spam email [103].

Targeting the most popular micro-blogging site with 200 million users, spammers have launched various attacks on Twitter, including spreading malware, phishing and scams. For example, Koobface [37], a worm targeting OSNs, broke out in 2009. It has several variants infecting Twitter along with other websites. Koobface spreads by delivering messages to cyber friends of a victim user whose computer has already been infected. And the message directs the recipients to a third-party website, where they are prompted to download what is purported to be an update of the Adobe Flash player. If the malware is executed, Koobface is able to infect those systems. Phishing is another security concern on Twitter. A wave of phishing attacks in early 2010 sent victim users private messages followed by a link to a fake Twitter login page, trying to steal account passwords. The attacks caused Twitter to re-design its private message system [53]. Moreover, Twitter is also glutted with unsolicited scam messages, such as advertising adult pills, credit debt consolidation and so on. According to Twitter's spam monitoring status [46], around 11% of messages on Twitter were spam in August 2009.

As a countermeasure, Twitter has released its rules against spam and abuse [56]. Accounts violating the rules will result in permanent suspension. The set of rules mainly define spam on Twitter in the following categories of content, behavior and social relationship. In the content category, it is forbidden to post content or URLs of any kinds of spam. Large numbers of unrelated @replies, mentions and #hashtags, or duplicate content are also disallowed. The behavior category covers both individual and collective behavioral codes. At the individual level, aggressive automation such as constantly running programs to post tweets without human participation is prohibited. At the collective level, using multiple accounts to post duplicate content is also considered as spamming. In terms of social relationship, one cannot follow a large number of users in a short amount of time, or have a small number of followers compared to the number of friends it is following, or create or purchase accounts in order to gain followers.

To avoid being detected by Twitter rules, social spammers have adopted a similar idea of email spam campaigns by coordinating multiple accounts to achieve a specific purpose. The spammer distributes the workload among spam accounts, thus individual accounts now may exhibit stealthy spam behavior and fly under the radar. Besides, multiple accounts also can spread spam to a wider audience. Some related studies have demonstrated the wide existence of spam campaigns on OSNs, such as Twitter and Facebook, respectively [92, 85]. The existing work mainly relies on the URL feature. More specifically, related messages with the shared final landing URL are clustered into a campaign. Then, the URL is looked up in URL blacklists. If the URL is blacklisted, the campaign is classified as a spam campaign; otherwise it is legitimate. Currently, the existing detection methods have some disadvantages listed as follows. First, URL blacklists have the lag effect, allowing more than 90% of visitors to click on a spam URL before it becomes blacklisted [92]. Furthermore, URL blacklists can only cover part of spam URLs, and thus some spam campaigns may escape

detection. Second, some URL blacklists generate false positive errors as they only check the host-name component of a URL, instead of the whole URL. For example, the URL shortening service `http://ow.ly` is listed on the URIBL blacklist [59] because it is greatly abused by spammers. Although `http://ow.ly/6eAci` is a benign URL that redirects to a CNN's report of Hurricane Irene, it is blacklisted by URIBL based on the hostname. Third, the URL feature generates false negative errors. For instance, consider a campaign that advertises a benign website in an aggressive spamming way. The spammer manipulates multiple accounts to post duplicate tweets about the website. The URL feature cannot classify the tweets as a spam campaign since the website URL is benign and not blacklisted. The first two disadvantages may be overcome by improving blacklisting process, but the third cannot be fixed by merely using the URL feature. Thus, the other features, such as collective posting content and behavior, should also be included. This chapter improves the existing work by introducing new features. The details of classification features are covered in Section 4.3.1.

There is existing work that addresses the social spam problem from a variety of aspects [86, 136, 139]. Thomas *et al.* [136] examined over one million spam accounts suspended by Twitter to characterize the behavior and lifetime of spam accounts. Their results confirm the appropriateness of classification features selected by this chapter, such as wide-spread abuse of URL shorteners, popular trends in hijacking and unsolicited mentions. Different from the analysis of [136] on spam accounts already classified by Twitter, our work designs a classification system to detect spam campaigns and involved accounts. Instead of building an independent spam filter for a social network, Wang *et al.* [139] proposed a framework for spam detection that can be used across all social network sites. The study of real datasets from several social networks demonstrates the feasibility of this experimental framework. Ghosh *et al.* [86] analyzed the strategies employed by Twitter spammers. By monitoring their link-creation strategies, they revealed that spammers adopt intelli-



gent “collaborative” strategies of link-formation to avoid detection and to increase the reach of their generated spam, such as forming “spam-farms” and creating a large number of links with targeted legitimate users.

### 4.1.3 Scope of This Chapter

A variety of spam attacks exist on Twitter. This chapter solely focuses on characterizing and detecting large-scale spam campaigns conducted on Twitter. The definition of spam in this chapter is spreading malicious, phishing or scam<sup>3</sup> content in tweets. Spammers may carry different purposes, but spam campaigns exhibit a shared feature that, they either create or compromise a large number of Twitter accounts to spread spam to a wide range of audience. Our work does not screen individual tweets to detect spam, and may miss small spam campaigns<sup>4</sup>. As a complement to existing spam detection methods, the main contribution of this chapter is detecting multiple related spam tweets and accounts in a robust and efficient way.

Note that after detecting a spam campaign, a site administrator may further classify the involved accounts into Sybil and compromised accounts, and process them accordingly. Here Sybil accounts refer to those created by spammers and exclusively used to post spam tweets. Compromised accounts refer to those used by legitimate users but hijacked by spammers to post spam without the observation of owners. Sybil accounts will be permanently suspended, while the owners of compromised accounts can be notified for spamming activities via their registration emails.

---

<sup>3</sup>We define a scam as any webpage that advertises a spectrum of solicitations, including but not limited to pornography, online gambling, fake pharmaceuticals.

<sup>4</sup>According to our clustering algorithm presented in Section 4.2.2, a single tweet may be clustered as a campaign if no other related tweets exist in the dataset.

## 4.2 Characterization

### 4.2.1 Data Collection

To measure the pervasiveness of spam, we conduct the data collection on Twitter from February to April in 2011. Twitter releases a set of APIs [138] that support large-scale data collection. Thanks to Twitter's courtesy of including our test accounts to its whitelist, our dataset accumulates more than 50 million tweets posted by around 22 million accounts. We develop a crawler in PHP which taps into Twitter's Streaming API [57] and Search API [60], respectively. The Streaming API outputs a small proportion of real-time global tweets via random sampling, and constitutes the majority of our dataset. The Search API enables the crawler running specific searches against the real-time index of recent tweets. As the privilege of our test accounts only allows the sampling rate of global tweets at 1%, the Search API gathers some complementary tweets according to our interest, such as those containing a specific URL. Since this work studies spam campaigns, we exclude tweets without URLs, and focus on the remaining 8 million tweets with URLs in the dataset. Due to the limited length of tweets, most spam tweets contain URLs to allure users to visit external spam websites. Thus, we assume that tweets without URLs are not spam. As shown in Section 4.2.2, our clustering algorithm is based on shared URLs.

URL redirection is widely used on Twitter. Normal users apply URL shortening services, such as t.co and bit.ly, to convert arbitrarily long URLs to short ones to better fit in tweets. Spammers also use shortening and other redirection techniques to hide original spam URLs and to avoid blacklist detection. We develop a Firefox extension in JavaScript to automatically visit every URL in the dataset and convert to its final landing URL if redirection is used. Some spammers tend to use long redirection chains that involve multiple hops (such as original URL -> intermediate URL -> ... ->

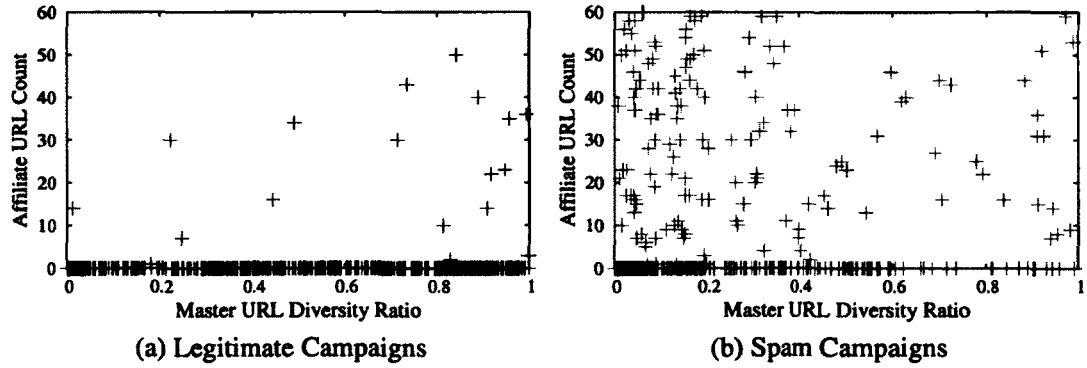


Figure 4.1: URL Statistics of Campaigns

final URL) to hide their traces. The extension records the whole chain, and provides a classification feature.

## 4.2.2 Clustering

We develop a clustering algorithm that clusters tweets into campaigns based on shared final URLs<sup>5</sup>. The idea behind the algorithm is that those tweets that share the same final URL are considered related. A tweet is modeled as the  $\langle \text{textual content, URL} \rangle$  pair. A given campaign,  $c_i$ , is denoted by a vector  $c_i = \langle u_i, T_i, A_i \rangle$ , where  $u_i$  is the shared final URL  $i$  for the campaign,  $T_i$  is the set of tweets containing  $u_i$ , and  $A_i$  is the set of accounts that have posted tweets in  $T_i$ . Let  $C$  denote the current set of campaigns. The clustering procedure iteratively chooses without replacement an arbitrary tweet  $t$  in the dataset. If the tweet's URL is  $u_{i'}$  and  $c_{i'} \in C$ , then the tweet is added in the campaign by updating  $T_{i'} = T_{i'} \cup \{t\}$ . If  $t$ 's account,  $a$ , is also new, then an update  $A_{i'} = A_{i'} \cup \{a\}$  is also performed. If  $c_{i'} \notin C$ , then a new campaign  $c_{i'}$  is created and  $C = C \cup \{c_{i'}\}$  is updated.

In our implementation, we store the dataset in MySQL database, and create a table for the clustering result. Every URL string is hashed, and the hash value is set as the table index. Two URL

<sup>5</sup>The subsequent campaign classification applies a variety of features, including both content and URL of tweets. More feature details are presented in Section 4.3.1.

strings are compared by their indexed hash values to improve the clustering performance. Once complete, the dataset includes 5,183,656 campaigns. The largest contains 7350 accounts with 9761 tweets posted.

### 4.2.3 Ground Truth Creation

After campaigns have been clustered, we need a ground truth set containing samples labeled as spam and legitimate campaigns. We can generate features from the two categories of labeled campaigns and develop a classification system. To create the ground truth, we choose some campaigns from our dataset<sup>6</sup>, manually perform several heuristics tests, and use human expertise to label unknown campaigns.

More specifically, we follow Twitter's spam rules during the manual inspection, and check both collective and individual features of an unknown campaign. First, we inspect the campaign's final URL. A batch script is performed to check the URL in five blacklists: Google Safe Browsing, PhishingTank, URIBL, SURBL and Spamhaus [31, 39, 59, 48, 45]. More details of the blacklist detection will be presented in Section 4.3.1. As we pointed out in Section 4.1.2, Google Safe Browsing and PhishingTank check the whole URL, while the other three only check the URL's hostname, which may generate false positives. Only if the URL is captured by the first two blacklists, is the related campaign directly labeled as spam without further manual inspection required. Otherwise, the human inspector visits the final URL in the browser, and checks the webpage content.

Second, we check the tweet content of the campaign. The script extracts the textual content from all the tweets in the campaign. The human inspects the content to see if (1) it contains spam

---

<sup>6</sup>The campaigns with a large number of accounts and tweets are favored during the selection, as large campaigns carry abundant collective behavior characteristics.

information, (2) it is unrelated with the URL's web content (namely, the URL is misleading), (3) duplicate or similar content is posted via single or multiple accounts. In addition, we also check content-related Twitter properties. For example, we inspect if tweets contain unrelated #hashtags or use @mentions to deliver unsolicited information to others.

Third we check the automation degree exhibited in the campaign, as automation is a good indicator of spam. The script presents the human inspector with the posting device makeup, the medium, and the entropy value of the posting inter-arrival timing sequence. The formal description of these features will be detailed in Section 4.3.1. Aggressive automation may raise the red flag, and influence the human's classification decision for the campaign.

By taking all of the above into consideration, the human inspector reaches the decision to label the campaign as spam or legitimate. In practice, we find out that most spam campaigns carry obvious characteristics of URL and content, making it easy to differentiate them from legitimate campaigns. We acknowledge that we may make mistakes in labeling campaigns, but believe that the error rate is very low. Finally, the ground truth set contains 744 spam campaigns and 580 legitimate ones.

#### **4.2.4 Campaign Analysis**

We now examine the characteristics of spam campaigns and compare with legitimate ones. The data analysis leads to the formal definition of classification features in Section 4.3.1.

We first discuss using URL statistics to reveal account connection in the campaign. We have observed that accounts in a legitimate campaign are usually run by independent users, while those involved in a spam campaign are often controlled by the same spammer. The URL statistics can provide hints of account connection. For clarity, we first define two terms: *master URL* and *affiliate URL*. For a normal URL such as <http://biy.ly/5As4k3>, affiliate URLs with it can be created by

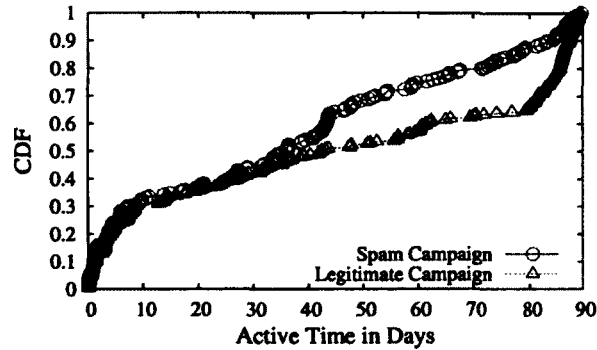


Figure 4.2: CDF of Campaign Active Time

appending random strings as the query component to the URL, such as <http://biy.ly/5As4k3?=-xd56> and <http://biy.ly/5As4k3?=7yfd>. The original URL is denoted as *master URL*. When the affiliate URL is clicked, the server still serves the web content of its master URL. Affiliate URLs help track the origin of click traffic. The spammer may use multiple accounts to advertise a spam URL. By assigning every account with a specific affiliate URL, the spammer can evaluate the spamming effect of individual accounts. This trick widely exists in online pyramid scams. Frequent appearance of affiliate URLs indicates strong connection among accounts. In contrast, different forms of master URLs indicate account independence. Although the tweets in a campaign share the same final URL, they may have different master URLs, such as <http://bit.ly/1wgYxU> and <http://ow.ly/6jRqX<sup>7</sup>>. Even if different users shorten the same final URL through a specific shortening service, they still get different forms of shortened URLs. For example, bit.ly may return two URLs for two users, <http://bit.ly/1wgYxU> and <http://bit.ly/3iMS15>, respectively. Moreover, the shortened URL contains a random string that is hardly possible to guess if the user was not informed. Thus, the pervasiveness of master URLs, to some extent, implies the account independence in a campaign. We define the master URL diversity ratio as the number of unique master URLs over the number of tweets in a

<sup>7</sup>All the URLs in this paragraph lead to <http://twitter.com>.

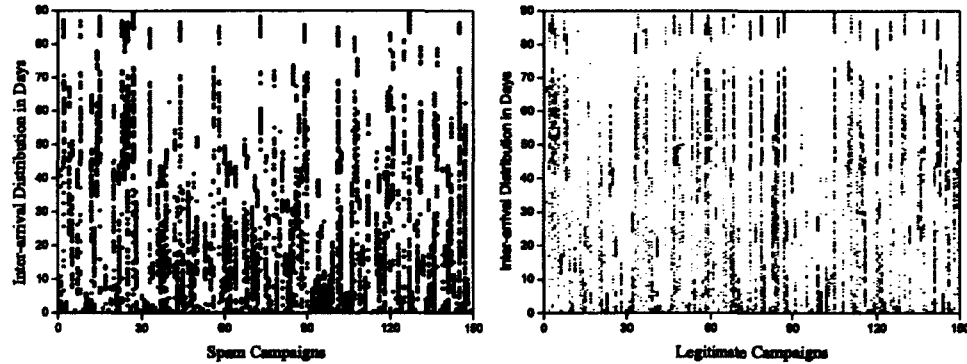


Figure 4.3: Inter-arrival Timing Distribution of Campaigns

campaign. A low ratio indicates the wide usage of affiliate URLs and account dependence, whereas a high ratio indicates the account independence. Figure 4.1 shows that more than 50% of spam campaigns use affiliate URLs, while only 3.6% of legitimate campaigns contain affiliate URLs. The average master URL diversity ratio of spam campaigns is 0.225, much lower than that of legitimate campaigns, at 0.423.

Now we analyze the temporal properties of campaigns. We define the active time of a campaign as the time span between its first and last tweet in our dataset. We point out a limitation of our dataset as our collection runs for three months while a campaign may exist before and/or after the measured period. While the largest possible active time in our dataset is 90 days, the actual time may be greater. Figure 4.2 shows the cumulative distribution function (CDF) of active time (in days) of spam and legitimate campaigns. Around 40% of campaigns in both categories have active time less than 30 days. For those longer than 30 days, the average active time of legitimate campaigns is 72.0 days, greater than that of spam campaigns at 59.5 days. Thanks to the workload distribution among accounts, the spamming behavior of an account may be stealthy during its initial stage, and avoid Twitter's detection. It explains the equal proportions of both categories within the 30-day time window. The accumulation of spamming behavior and the increase of campaign size expose spam

accounts, and many of them get suspended by Twitter. Beyond the 30-day window, the average active time of spam campaigns is clearly shorter than that of legitimate ones. However, more efforts need to be made to detect and eliminate spam campaigns in the initial stage for damage control.

The burstiness characterizes the overall workload distribution of spam campaigns. Figure 4.3 plots the inter-arrival timing pattern of two categories of campaigns. Due to space limit, each category contains 150 individual campaigns. Each campaign is represented by a vertical strip. In a campaign, tweets are sorted on timestamp, and the timestamp of the first tweet is set as 0. Each tweet corresponds to a tiny horizontal segment in the strip, and a block of intensive strips represent a burst of tweets in the campaign. A large number of spam campaigns show burstiness in the early stage. Some spammers aim to achieve the spamming goal in a quick way, and direct spam accounts to massively post tweets. Although the workload is distributed to multiple accounts, the inter-arrival timing pattern can still be used to reflect the overall workload of the campaign. The gradual suspension of spam accounts causes the stagnation in the late stage. Many legitimate campaigns tend to take a while to grow up, and demonstrate burstiness in the late stage. A popular legitimate campaign generates the epidemic effect by making more users tweet about it, spreading to even the larger audience.

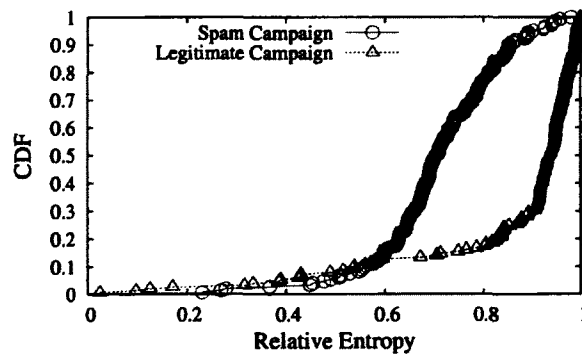


Figure 4.4: CDF of Entropy of Campaign Posting Inter-arrivals



Entropy is another temporal property that detects periodic or regular timing of posting patterns in a campaign. As the campaign contains multiple tweets carrying individual timestamps, we sort the timestamps and generate the inter-arrival sequence of posting timing. In Information Theory, the entropy rate is a measure of the complexity of a random process [79]. A high entropy rate indicates a random process, whereas a low entropy rate indicates a regular one. As the inter-arrival sequence of the campaign is not infinite, we use *corrected conditional entropy* to approximate the entropy rate. More theoretical proofs can be found in [78]. The significance of entropy does not exist in its absolute numeric value, but relative tendency. To get relative entropy for every campaign, we normalize entropy values via dividing them by the maximum value of the campaign in the ground truth set. Figure 4.4 plots the CDF of relative entropy of posting inter-arrivals of both categories. Two curves interweave in the range between  $[0, 0.6]$ . The behavior of auto programs (namely Twitter bots) is often less complicated than that of humans, which can be measured by low entropy rate. A part of both spam and legitimate campaigns use auto programs to post, causing the low entropy. However, in the range between  $[0.6, 1]$ , the relative entropy of the legitimate category is clearly higher than that of the spam category. The majority of spam campaigns (and a large proportion of their accounts) run auto devices to post, driven by regular or pseudo-random timers. In contrast, tweets in legitimate campaigns are mostly posted humans. The intrinsic irregularity and complexity of human behavior generates a higher entropy rate for legitimate campaigns. We also find an interesting fact that, a small part of spam campaigns post their tweets manually, generating high entropy. We speculate it is either a form of click farm on Twitter, or some spammers are not professional, and do not know how to run auto programs to tweet.

Now we discuss a dilemma spammers often face, namely reusing spam accounts. If multiple tweets in the campaign are posted by an account, considering the tweets share the same final URL,

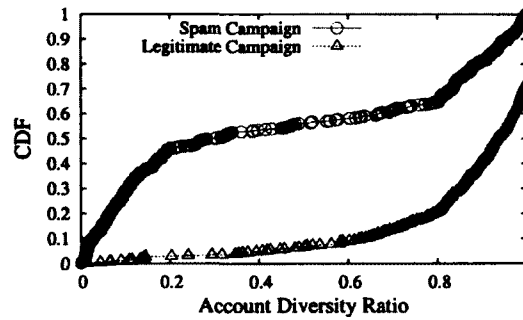


Figure 4.5: CDF of Account Diversity Ratio of Campaigns

the account exhibits the evidence of duplicated posting, which is an indicator of spam. We introduce the account diversity ratio feature. For normalization, this feature is defined as the number of accounts in the campaign over that of tweets. Figure 4.5 plots the CDF of this feature of both categories. Spammers want to operate accounts in a stealthy way, which requires individual accounts to post few tweets. In reality, it costs effort to get followers to a spam account, and the number of “influential” accounts owned by a spammer is limited. Thus, the spammer tends to repeatedly use accounts to post duplicate spam, causing the low ratio. The figure clearly demonstrates that, the account diversity ratio of legitimate campaigns is much higher than that of spam campaigns. In particular, about 28.8% legitimate campaigns have the ratio as 1, meaning every tweet in the campaign is posted by a unique account. The average ratio of legitimate campaigns is 86.4%, while that of spam campaigns is 45.0%. It further suggests that, legitimate campaigns have stronger account independence than spam campaigns.

### 4.3 Design

In this section, we first present the design philosophy of the classification system. In particular, we formally describe classification features and introduce semantic similarity to detect duplicate

content in a campaign. Then, we implement the classifier based on the Random Forest algorithm.

### 4.3.1 Classification Features

The classification involves a variety of features, ranging from individual tweet/account levels to a collective campaign level. No single feature is capable of discriminating effectively between spam and legitimate campaigns. Here we introduce these features used in our classification, and later the machine learning algorithm will decide the importance (namely weight) of the features during the training, which is shown in Section 4.4.1.

#### 4.3.1.1 Tweet-level Features

We start with tweet-level features, as tweets are the atomic unit of Twitter. A tweet is modeled as the <textual content, original URL> pair.

**Spam Content Proportion.** Some spam tweets carry explicit spam information, such as “buy Viagra online without a prescription” and “get car loan with bad credit”. We create a list of spam words with high frequency on Twitter to capture spam content. The tweet text is tokenized into words which are further checked in the spam word list. This feature is defined as the number of spam words over the total word number in a tweet .

**URL Redirection.** We develop a Firefox extension to check the original URL in the tweet. If URL redirection is used, it records the final landing URL. By recording the status change in the browser’s address bar, the extension logs the whole redirection chain (such as original URL -> intermediate URL -> ... -> final URL). Besides the binary redirection flag, hop number also serves as a useful feature. Spammers tend to use multi-hop redirection to hide spam origins and avoid URL blacklists.

**URL Blacklisting.** We check the final URL in five blacklists including Google Safe Browsing, PhishingTank, URIBL, SURBL, and Spamhaus [31, 39, 59, 48, 45]. Google Safe Browsing checks URLs against Google’s constantly updated lists of suspected phishing and malware pages. *PhishingTank* focuses on phishing websites. The mechanisms of URIBL, SURBL and Spamhaus are similar. They contain suspicious websites that have appeared in spam emails, primarily Unsolicited Bulk/Commercial Email (UBE/UCE). For the first two blacklists, we download them to our database, and do the local lookup. For the remaining three lists, we inquire their servers by issuing DNS queries. If the URL appears in any of the blacklists, the feature is set as true. As the tweets in a campaign share the same final URL, this operation only needs to be performed once.

#### 4.3.1.2 Account-level Features

We also collect data of Twitter accounts involved in a campaign by calling Twitter’s REST API [55], and present account-level features to characterize accounts.

**Account Profile.** An account has a self-introduction profile consisting of a short description text and homepage URL. We check whether the description contains spam or the URL is blacklisted.

**Social Relationship.** Tweets of an account can only be delivered to its followers. To achieve a wide influence, the spammer needs to accumulate a large number of followers. However, normal users are unlikely to follow spam accounts. A common trick shared by spammers is following a great number of users (either targeted or randomly selected), and expecting some of them to follow back. Following back the friend request is considered as the etiquette on Twitter. Many spam victims blindly follow back “spammer friends” without carefully checking those suspicious accounts. For an account, we calculate its friend count, follower count, and the ratio between them.

**Account Reputation.** Extended from the previous feature, we have observed that users are

likely to follow “famous” accounts. This feature is calculated and normalized as  $\text{follower count} / (\text{follower count} + \text{friend count})$ . A celebrity usually has many followers and few friends<sup>8</sup>, and its reputation is close to 1. However, for a spammer with few followers and many friends, its reputation is close to 0.

**Account Taste.** Intuitively, the account chooses whom to follow (namely, friends), and this reflects its “taste”. If it follows spammers, its “taste” is bad. By doing this, it helps spread spam to more users, making itself a “supporter” of spammers. This feature is defined as average *Account Reputation* of all the friends of the account.

**Lifetime Tweet Number.** Spam accounts may get suspended for aggressively posting spam. Due to the short lifetime, averagely spam accounts may post fewer tweets. This feature shows the number of tweets an account has posted in lifetime when it is visited by our crawler.

**Account Registration Date.** Spammers may frequently create new accounts to replace suspended ones. Many spam accounts in our measurement have been created recently.

**Account Verification.** Twitter verifies accounts for celebrities and organizations. It is difficult for spammers to acquire verified accounts. This binary feature shows whether the account is verified or not.

**Account Protection.** For user privacy, an account that opts in the protection option makes its tweets invisible to general public, and only visible to approved followers. The option conflicts with the purpose of spreading spam to the wide audience, and may not be adopted by spam accounts.

---

<sup>8</sup>For example, @Yankees, the official Twitter account of New York Yankees, has 400,000 followers and only 29 friends.

### 4.3.1.3 Campaign-level Features

Collective features may reveal the characteristics of spam campaigns that cannot be observed through individual features. At last we present the campaign-level features as follows. The features of the account diversity ratio, the original URL diversity ratio, the affiliate link number and the entropy of inter-arrival timing have been explained in Section 4.2.4.

**Hashtag Ratio.** Spammers often hijack trending hashtags and append them to unrelated spam tweets to increase the chance of being searched and displayed. The feature is defined as the number of hashtags in the tweets over the number of tweets of the campaign.

**Mention Ratio.** Another trick spammers often play is using @mention to deliver spam to targeted users even without the existing social relationship. The feature is defined as the number of mentions in the tweets over the number of tweets of the campaign.

**Content Self-similarity Score.** A spam campaign may contain similar tweets created by spam content templates. Users in a legitimate campaign usually contribute content individually, and may not show a strong self-similarity. This feature measures the content self-similarity of the campaign. The details are presented in Section 4.3.2.

**Posting Device Makeup.** Twitter supports a variety of channels to post tweets, such as web, mobile devices, and 3rd-party tools. The 8 million tweets in our campaign dataset are posted by 44,545 distinct devices. In the perspective of behavior automation, they can be divided into two categories: manual and auto devices. Manual devices require direct human participation, such as tweeting via web browser or smart-phone. Auto devices are piloted programs that automatically perform tasks on Twitter, and require minimum human participation (such as importing Twitter account information). We manually label the top 100 devices as manual or auto, and use the tdash's

API to process the rest. The tdash website monitors the statistics of Twitter devices by maintaining a database of 132 million indexed tweets and 159,000 detected devices [49]. In the campaign dataset, around 62.7% of tweets are posted by manual devices, and the rest 37.3% by auto devices. For every campaign, the script checks its posting devices against the labeled device list, and calculates the proportions of manual and auto devices as the value of posting device makeup.

### 4.3.2 Content Semantic Similarity

Spammers may use content templates to create similar spam tweets. Calculating semantic similarity can detect duplicate or similar content in multiple tweets in the campaign. The calculation is challenging as short messages like tweets do not carry as many semantic features as long texts (i.e. email bodies). The traditional similarity measures, such as text shringling based on hashing, do not work well in the context of Twitter. Our work applies the Vector Space Model [125] that converts tweet texts into vectors, and then calculates the cosine similarity between them. Equation 4.1 denotes the cosine similarity between two  $n$ -dimensional vectors,  $A$  and  $B$ .

$$\cos\_sim = \frac{A \bullet B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n A_i^2} \times \sqrt{\sum_{i=1}^n B_i^2}} \quad (4.1)$$

For implementation, we use SenseClusters, an open-source program [43], that clusters text messages based on contextual similarity. Given the set of tweets in the campaign, we treat it as a text corpus, and generate a vocabulary by extracting distinct words from the corpus. Then we generate an occurrence matrix with tweets as rows, and words in the vocabulary as columns. The value of  $cell_{ij}$  is the TF-IDF (Term Frequency - Inverse Document Frequency) weight [65], which represents the occurrence frequency of  $word_j$  in  $tweet_i$ . The TF-IDF weight is a statistical measure

that evaluates a word's importance to a text in the corpus. The importance increases proportionally to the number of times the word appears in the text, but is offset by the frequency of the word in the corpus. As the occurrence matrix may be very sparse, Latent Semantic Analysis (LSA) and Singular Value Decomposition (SVD) are performed for rank lowering while preserving the similarity structure among rows [83, 80]. As the most intuitive approach, 1st-order similarity detects the number of exact words shared (or overlapped) between tweets. Because spam templates often adopt synonym interchanging for the purpose of obfuscation, our work applies 2nd-order similarity to measure similar tweets. Its general idea is to replace the context with something else that will still represent it, and yet likely provide more information from which similarity judgments can be made [121]. SenseClusters uses WordNet, a lexical database of English [113], for word replacement and expansion. Given the tweet corpus, SenseClusters divides  $N$  tweets into  $K$  clusters based on the semantic sense on the fly.

**Table 4.1:** A Clustering Example of Semantic Similarity

Cluster	Size Proportion	Similarity Score
1	10%	0.9
2	30%	0.7
3	60%	0.1

We present Table 4.1 as an example for the illustrative purpose. The tweet corpus is partitioned into three clusters, and a similarity score is assigned to each cluster to evaluate how similar the tweets it contains are. The semantic topics of Clusters 1 and 2 are credit card debt and music video, respectively. The remaining unrelated tweets are assigned to Cluster 3.

#### **Cluster 1.**

- How to consolidate credit card debt
- Consolidate credit cards now to become debt free later



- ...

#### Cluster 2.

- Amazing this music footage you'll like
- This awesome music video hope u like
- ...

#### Cluster 3.

- The remaining unrelated tweets

We design Equation 4.2 to measure the self-similarity of the campaign's tweet content.

$$self\_sim\_score = \sum_{i=1}^K \frac{cluster_i\_size^{w1} * cluster_i\_sim^{w2}}{K^{w3}}, \quad (4.2)$$

where  $K$  is the number of semantic clusters in the campaign, and  $w1$  to  $w3$  are weight factors with their tuning procedure presented in Section 4.4.1.

### 4.3.3 Machine Learning Classifier

Our classification problem can be defined as follows. Given a campaign,  $c = \langle u, T, A \rangle$ , the classifier determines  $c$  as a either spam or legitimate campaign. We choose Random Forest [70] as the machine learning algorithm<sup>9</sup>, and train the classifier to make the binary decision. Random Forest serves as an ensemble classifier that includes multiple decision trees. The algorithm combines the bagging idea in [70] and random feature selection in [95] to construct a “forest” of decision trees with controlled variation. Suppose the training set contains  $M$  features, and each decision tree only

---

<sup>9</sup>The reason is explained in Section 4.4.2

uses  $m(\ll M)$  features to reach the decision. The decision tree consists of two types of nodes, the leaf node labeled with the class, and the interior node that corresponds to a feature and links to a subtree. The tree is built from the root down to leaves in a recursive manner. During the construction path, every interior node is associated with the feature that is most informative among the remaining features not yet included in the path. Entropy is used to measure the information gain (namely, how informative a feature is). For classification, an unknown sample is pushed down the tree, and assigned with the class of the leaf node where the sample ends up. More details about decision tree can be found in [105]. Given a specific sample, every decision tree makes a classification decision (either spam or legitimate campaign in our case), and Random Forest applies the majority voting of all the trees to reach the final decision.

## 4.4 Evaluation

In this section, we first train the classifier. Then, we evaluate the accuracy of our classification system based on the ground truth set.

### 4.4.1 Training

As described in Section 4.2.3, our ground truth set consists of manually labeled campaigns. More specifically, 744 spam campaigns contain around 70,000 accounts and 131,000 tweets, whereas 580 legitimate campaigns contain around 150,000 accounts and 180,000 tweets.

Before training the classifier, we need to determine the content self-similarity feature by tuning the weight factors in Equation 4.2 with the following method. We choose Decision Tree as the tuner, and the feature represented by the self-similarity score as the only classification feature. We

**Table 4.2: Algorithm Performance Comparison**

Feature	Accuracy (%)	FPR (%)	FNR (%)
Random Forest	94.5	4.1	6.6
Decision Table	92.1	6.7	8.8
Random Tree	91.4	9.1	8.2
KStar	90.2	7.9	11.3
Bayes Net	88.8	9.6	12.4
SMO	85.2	11.2	17.6
Simple Logistic	84.0	10.4	20.4
J48	82.8	15.2	18.8

try different combinations of numeric values of  $w_1$  to  $w_3$ . In every test round, a combination generates a different self-similarity score for a campaign in the ground truth set. The decision tree associates the self-similarity feature with the root<sup>10</sup>, and calculates the best split between spam and legitimate campaigns. The combination of ( $w_1 = 0.8, w_2 = 0.5, w_3 = 1$ ) generates the highest overall accuracy on the ground truth set, and is chosen for Equation 4.2. As a campaign may be partitioned into a few semantic clusters, the cluster size proportion is no greater than 1. According to the algorithm of the SenseClusters, the cluster similarity score assigned is also no greater than 1. Note that  $w_1$  and  $w_2$  are decimal fractions, and they add more weights to the cluster size and cluster similarity. Furthermore,  $w_2$  makes cluster similarity more important than cluster size, as  $w_2$  is less than  $w_1$ .

#### 4.4.2 Cross Validation

By calculating the values of the features described in Section 4.3.1, a feature vector is generated for each campaign. Weka supports a collection of machine learning algorithms for classification, including mainstream categories of Bayes, trees and so on [93]. We try multiple algorithms in each category, list and compare performance results for the top classifiers with accuracy greater than 80%

<sup>10</sup>As it is the only feature in the classification.

in Table 4.2. For each classifier, we use Cross Validation with ten folds to train and test it over the ground truth set [111]. The dataset is randomly partitioned into ten complementary subsets with equal size. In each round, one out of ten subsets is retained as the test set to validate the classifier, while the remaining nine subsets are used as the training set to train the classifier. At the beginning of a round, the classifier is reset and re-trained. Thus, each round is an independent classification procedure, and does not affect subsequent ones. The individual results from ten rounds are averaged to generate the final estimation. The advantage of cross validation is that, all samples in the dataset are used for both training and validation, while each sample is validated exactly once.

Table 4.2 lists three metrics for evaluating the classification performance sorted on accuracy. Considering the confusion matrix with spam campaigns as positive cases, *Accuracy* is the proportion of samples that are correctly identified, *False Positive Rate* (FPR) is the proportion of negatives cases that are incorrectly classified as positive, and *False Negative Rate* (FNR) is the proportion of positives cases that are incorrectly classified as negative. During evaluation, we expect to constrain the FPR low at the cost of accepting the medium FNR. Classifying benign campaigns as spam upsets legitimate users, while missing a small part of spam campaigns is tolerable. Random Forest achieves the highest accuracy, lowest FPR and FNR, and hence is selected as the final classifier for our dataset.

Some features play a more important role than others during the classification. Subsequently, we attempt to evaluate the discrimination weight each feature has. Similar to the tuning method for Equation 4.2, in each test, we use only one feature to independently cross validate the ground truth set with Decision Tree<sup>11</sup>. The one with the highest accuracy may be considered as the most important feature. Table 4.3 presents the performance results of the top 10 features, which are also

---

<sup>11</sup>Random Forest transforms to Decision Tree in the case of single-feature classification.

sorted on accuracy. The Account Diversity Ratio feature has the highest accuracy at 85.6%. Technically this one is not difficult to bypass, because spammers could use a large amount of accounts to distribute the workload and lower the ratio. However, spam accounts with limited normal followers cannot generate the satisfying propaganda. We speculate that, in reality, spammers tend to repeatedly use “influential” accounts to deliver spam to a wide audience. The Timing Entropy feature captures the intrinsic complexity of human behavior, that is difficult for bot accounts to bypass. However, many spam campaigns involve manual accounts (probably in the form of click farm), that generate the high FNR at 22.8% for the feature.

We are particularly interested in the performance of the URL Blacklists feature, as it is used as the only feature for spam campaign detection in some existing work. We present the performance comparison between our Random-Forest-based classifier that applies multiple features and the single blacklist feature. Blacklists are haunted by the inevitable lag effect, and cannot include all spam sites “in-the-wild”. Besides, blacklists cannot detect duplicate spamming over multiple accounts. These factors generate a high FNR at 29.0%. By using multi-dimensional features, our classifier manages to capture more spam campaigns that would have been missed by the blacklist feature, and lowers the FNR at 6.6%. The low FPR of the blacklist feature is caused by the fact that, some blacklists only check the hostname of URL, and mis-classify some benign web pages hosted by blacklisted websites. The FPR of our classifier (4.1%) is slightly higher than that of the blacklist feature (3.2%). Considering the big gain on FNR, we think the FPR cost is acceptable. Overall, our classifier improves the accuracy from 82.3% to 94.5%.

Lastly, two content-related features, Content Self-similarity and Spam Word Ratio, sink in the bottom of the table due to the following reasons. First, a proportion of spam tweets deliver spam via the combination of innocent content and spam URLs. Second, spammers frequently change and

Table 4.3: Feature Performance Comparison

Feature	Accuracy (%)	FPR (%)	FNR (%)
Account Diversity Ratio	85.6	16.2	13.0
Timing Entropy	83.0	9.5	22.8
URL Blacklists	82.3	3.2	29.0
Avg Account Reputation	78.5	25.6	18.3
Active Time	77.0	16.2	28.3
Affiliate URL No	76.7	9.6	34.0
Manual Device %	74.8	10.3	36.8
Tweet No	75.4	28.6	21.5
Content Self Similarity	72.3	33.7	23.0
Spam Word Ratio	70.5	25.8	32.4

obfuscate spam words, and correspondingly, detectors need to keep the spam word list updated. Finally, limited by the text length, tweets do not carry abundant semantic characteristics. The application of natural language processing in the further research could improve the detection accuracy of spam patterns and content similarity.

#### 4.4.3 System Overhead

We run experiments on a workstation with an Inter Core 2 Duo 2.4 GHz CPU and 4 GB memory. The Random Forest classifier only costs 0.12 seconds for the cross validation over the ground truth set. Thus, the computational overhead is negligible.

## 4.5 Conclusion

Spam haunts social networks, as social relationship facilitate spam spreading. Conventional spam detection methods check individual accounts or messages for the existence of spam. Our work switches to the collective detection perspective by capturing spam campaigns involving multiple accounts. Our work uses features combining both content and behavior to distinguish spam cam-

paings from legitimate ones, and build an automatic classification framework. Our work can be applied to other social networks by integrating application-specific features.

Detecting spam is an endless cat-and-mouse game, so the idea of building-once-and-using-forever is not realistic. The classifier is built with a specific set of features. As spamming methods may evolve in the future, some features may be added or replaced with new ones. The classifier should also be re-trained with the up-to-date ground truth dataset.

## Chapter 5

Blogs (from *weblog*), are a popular application of Web 2.0. Internet users publish articles on blog sites, such as personal online diaries or news on a particular subject. Like normal web pages, blog pages are primarily textual combined with images, videos, and links. The distinctive feature of blog is user interaction, which allows visitors to leave comments to blog articles. A visitor fills in the comment form and submits it, and his comment will display below the article in reverse-chronological order. Unfortunately, the increasing popularity of blogs and the simplicity of posting comments have made it easy for blog bots to automatically post comments with malicious intent. According to the estimation of [44], about 83 percent of blog comments are injected by blog bots, indicating how rampant blog bots are in the blogosphere. Most of these automated comments are associated with spam websites, containing either traceback links to inflate search engine rankings [104], or other content to lure visitors to these sites.

Since the majority of content generated by blog bots is unwanted by blog owners and visitors, blogging software has incorporated a variety of methods to discourage posting from these sources. Fundamentally, detecting human presence is an effective defense against blog bots. Conventional detection methods based on Human Interactive Proofs (HIPs) [75] usually require direct participation from human users, such as CAPTCHA. As a reverse Turing test, it challenges a user with an image carrying alphanumeric text. The user must enter the exact text before the blog site can accept the comment for submission. To cope with an advanced bot's capability for image recognition



(namely, De-CAPTCHA), CAPTCHA tools add image noise to the background canvas, and greatly distort characters [146]. However, such a CAPTCHA validation also requires non-trivial effort from human users. In some cases, users have to try several times to correctly recognize a CAPTCHA image because it has become more and more difficult even for human to read. Such a validation in place may effect a significant decrease in participation from human visitors.

In this chapter, we present a new method based on passive monitoring for blog bot detection, as conventional detection systems have become a nuisance for human users. Our proposed approach employs behavioral biometrics, including mouse and keystroke dynamics, to distinguish between human and bot. It has two major advantages over existing solutions. First, it uses continuous monitoring throughout the entire user session, and eliminates single checkpoints. In contrast, blog sites deployed with the conventional detection face the dilemma of applying one-time test or multiple tests. On one hand, blog bots can pass the one-time test, such as account login, with the help of human. On the other hand, multiple tests, such as recognizing a CAPTCHA image before each comment posting, are too intrusive for human users. Our passive continuous monitoring resolves the above dilemma. Second, our method is non-interactive and completely transparent to users. Moreover, no detection decision needs to be made until the user submits the comment, which in turn saves system resources. We develop a passive, webpage-embedded logger to collect user input activities on a real, active blog site. By measuring and characterizing biometric features of user input data, we discover the fundamental differences between human and blog bot in how they surf web pages and post comments. These results greatly facilitate accurate detection of blog bots.

We build a prototype of an automatic classification system that detects blog bots based on user input data. The system consists of two components, a webpage-embedded logger and a server-side detector. The logger is implemented as a JavaScript snippet that runs in the webpage on the

client browser. It records a user's input actions during her stay at the site and streams the data to the server-side detector. The detector processes raw user input (UI) data, and extracts biometrics-related features. The core of the detector is a machine-learning-based classifier which is tuned with training data for the binary classification, namely determining whether the user is human or bot. Informed with the classification result, the server decides whether or not to accept the comment form submission<sup>1</sup>. We evaluate the efficacy of the detection system by conducting a series of experiments over the user input dataset. The experimental results demonstrate that the system can detect 97.9% of current blog bots with extremely low false positive rate of 0.2%.

As defense against bots is a challenging task, we acknowledge that our detection alone cannot eliminate the problem. However, our approach is a significant complement to conventional HIPs. We believe that, with the inherent irregularity and complexity of human behavior, it is extremely difficult if not impossible for a bot to completely mimic human behavior. Our behavior-based detection raises the bar for bot participation during this game of cat-and-mouse.

The remainder of the chapter is organized as follows. Section 5.1 covers related work on blog bot and behavioral biometrics. Section 5.2 details our measurements and characterization of user inputs from human visitors and blog bots, respectively. Section 5.3 describes our automatic classification system. Section 5.4 evaluates the system efficacy for detecting blog bots. Section 5.5 discusses potential evasion against our detection system. Finally, Section 5.6 concludes the chapter.

---

<sup>1</sup>For instance, the server can be configured to accept the manual submission from human, and reject the automated form completion from bot.

## 5.1 Background and Related Work

From the perspective of blog content creation, there are two types of blog bots. The first type is the article posting bot, which automatically publishes blog articles. For example, it pipelines RSS feeds from other sites as articles into the blog site, or posts preset content for a spam blog (also known as a splog). Since the posting of articles usually requires the elevated privilege of the webmaster, article posting bots are not the focus of this study. The second type is the comment posting bot, which posts comments or replies to blog sites. Given a link to a blog site, this bot analyzes the HTML structure of the blog article, especially the “leave a comment” form, fills in input fields, and posts a comment automatically. Most blog sites do not require visitors to register to post comments, and thus give ample space for bots to exploit. The focus of our work is on this bot type, and the term “blog bot” in the remainder of the chapter implicitly refers to comment posting bots. Currently, blog bots are mainly created to fulfill two tasks. First, the bot posts a comment with a backlink directing to a specific website (such as that of the bot owner)<sup>2</sup>. Posting backlinks to numerous blog sites has the effect of increasing the search engine traffic, in an attempt to boost search rankings for the originating site. The search industry has already employed some mitigation measures, such as Google’s no-follow tag to prevent spam from polluting on search rankings. However, bots still massively generate inflation backlinks due to the ease and low cost of posting. Second, bots post comments with spam content (also known as spam comments) aiming to lure visitors to spam-related or other malicious sites<sup>3</sup>. Many blog sites eliminate spam comments based on content filtering, and Akismet [4] is such a distributed anti-spam web service. Each time

---

<sup>2</sup>Here is an example of backlink comment, “*I don’t really think this is right, but believe whatever you want. The real story can be found here on my blog: <http://myblog.com/blog>”.*

<sup>3</sup>Here is an example of spam comment, “*Thousands of cheap replica watches and fashionable designer bands at [www.hot-replica.com/](http://www.hot-replica.com/)”.*

a new comment is posted to the blog, it is submitted to Akismet, which checks content, runs other tests, and returns the spam detection result to the blog. Our work has the different research direction, and checks posting behavior instead of content posted.

### 5.1.1 Existing Web Bot Detection

There have been many previous works on web bot detection. Stassopoulou et al. [128] introduced a probabilistic modeling approach for web bot detection by analyzing server access logs. They constructed a Bayesian network that classifies log sessions as being crawler or human induced. Their classifier uses some features to characterize crawler and human behaviors, including maximum sustained click rate, session duration, percentage of image requests, pdf/ps requests, 4xx error responses, and robots.txt file requests. Tan et al. [134] did a similar study by investigating navigational patterns of web bots. They extracted features from server logs such as total number of pages requested, average time between two HTML requests, and percentage of requests made with GET/POST methods, which are useful for a machine learning algorithm to distinguish between human and bot. Park et al. [120] took web bot detection as a special form of the Turing test and defended the system by inferring whether the traffic source is human or bot. More specifically, their detection decision depends on evidence of mouse movement or keyboard activity from the client. If no human input activities are detected by a web server, the user is classified as bot.

However, all these existing detection mechanisms are only effective for detecting form-injection bots that do not generate any human-like activities. They miss more advanced bots such as human-mimic bots, which can navigate web pages in the browser by generating simple user input actions. A web server cannot easily detect this type of bot by using navigational patterns or logs. Moreover, since human-mimic bots send mouse and keystroke actions to the browser, the detector in [120] is

deceived by the existence of forged human activity. Different from the previous detection research, our proposed approach does not merely depend on the presence of mouse or keystroke actions to distinguish human from bot. Our solution extracts features from UI actions that represent the inherent irregularity of human behaviors, and applies these features for bot detection.

### 5.1.2 Behavioral Biometrics

The fundamental idea of our approach is to exploit behavioral biometrics for bot detection. Biometrics-based authentication is defined as the automated use of a collection of factors describing human behavioral or physiological characteristics to establish or verify a precise identity [110]. It can be classified into two categories: physiological biometrics and behavioral biometrics. Physiological biometrics uses measurements from the human body, including fingerprints, iris, retina, and facial scanning, and so on. Behavioral biometrics uses measurements based on human actions, such as signatures, voice and keystroke dynamics. Compared with physiological biometrics, normally behavioral biometrics do not require any special-purposed hardware for data collection, and are easy to employ.

Among all behavioral biometrics, mouse and keystroke dynamics are the most common metrics attempted for on-line user authentication [68] [115] [64]. Keystroke dynamics measures duration (the length of time a key is pressed down) and inter-arrival time (the time from pressing one key to another) for keystroke actions. In the previous works [68], [71], [115], the way that a user types at the keyboard is analyzed to identify its habitual typing rhythm and patterns. Mouse dynamics measures the characteristics of mouse actions of an individual user when it is interacting with the graphical user interface (GUI). Raw events generated by the mouse input device include cursor movement, mouse button press and release. In [64], high-level mouse actions are defined as the

four meaningful combinations of raw events: Mouse-Move (i.e., general mouse movement), Drag-and-Drop (i.e., the action starts with mouse button down, movement, and then button released), Point-and-Click (i.e., mouse movement followed by a click or a double click), and Silence (i.e., no movement). Using neural networks, Ahmed et al. [64] modeled the mouse dynamics characteristics from the captured user input data. They implemented a detector that generates a signature for a user. User identification is conducted by comparing two signatures.

Our chapter extends behavioral biometrics into blog bot detection, mainly using keystroke and mouse dynamics. In the context of blog user behavior characterization, keystroke and mouse dynamics are complementary to each other. This is because human users move the mouse cursor to surf blog pages, and strike the keyboard to post comments. However, our work significantly differs from aforementioned biometric detectors. Our work distinguishes two classes of users, human and blog bot, instead of identifying individual users. Some previous work indicates that, behavioral biometrics may generate non-negligible errors in identifying individuals as one's behavior may vary significantly [66]. Our evaluation results demonstrate behavioral biometrics works accurately for the problem of classifying two classes: bot and human, instead of user identification.

The closest previous work to ours is [88], which also applies behavioral biometrics for detecting game bots in online games. On one hand, blog bot detection is different from game bot detection, due to different application environments<sup>4</sup>. In [88], the user input actions are collected by the game client, while our work resorts to JavaScript in the blog page for user input collection. Neural networks are used in [88], while our work uses decision tree for classification. Decision tree is more efficient than neural networks on our dataset of user input activities, and the tree structure clearly presents how features are weighted during the classification. On the other hand, behavioral

---

<sup>4</sup>Game bot operates on a map and fulfills a series of game-related missions.

biometrics works well for both cases, and user input behaviors remain consistent in different online applications, either online blogging or online gaming.

## 5.2 Behavior Characterization

In this section, we analyze user behaviors, namely how a user surfs blog pages and posts comments, based on data collected from a large corpus of users. We first introduce three types of blog bots, then describe how we collect user input data from a blog site. Finally, we characterize the behavioral differences between human and blog bot, in terms of keystroke and mouse dynamics.

**Table 5.1: User Input Actions**

Action	Description
Keystroke	The press and release of the same key
Point	A set of continuous mouse moves with no mouse clicks, and the interval between two consecutive moves is no more than 0.4 second
Click	The press and release of the same mouse button
Point-and-Click	A point followed by a click within 0.4 second
Drag-and-Drop	Mouse button down, movement, and then mouse button up

### 5.2.1 Blog Bots

Fundamentally, current blog bots can be categorized into three different types based on their working mechanisms: Form Injection Bot, Human Mimic Bot, and Replay Bot. Form Injection Bots do not post comments via the browser. Rather, it directly sends an HTTP request to the server for the blog page where it plans to post comments. After receiving the HTML content of the requested page, it analyzes the HTML structure of the comment form. Then, it injects content into form fields<sup>5</sup>,

<sup>5</sup>The form is usually well-structured, and the ID/name of each input field remains constant. For example, `<input type="text" name="email" />` is the text field to enter email address. Thus, the bot author programs the bot to recognize

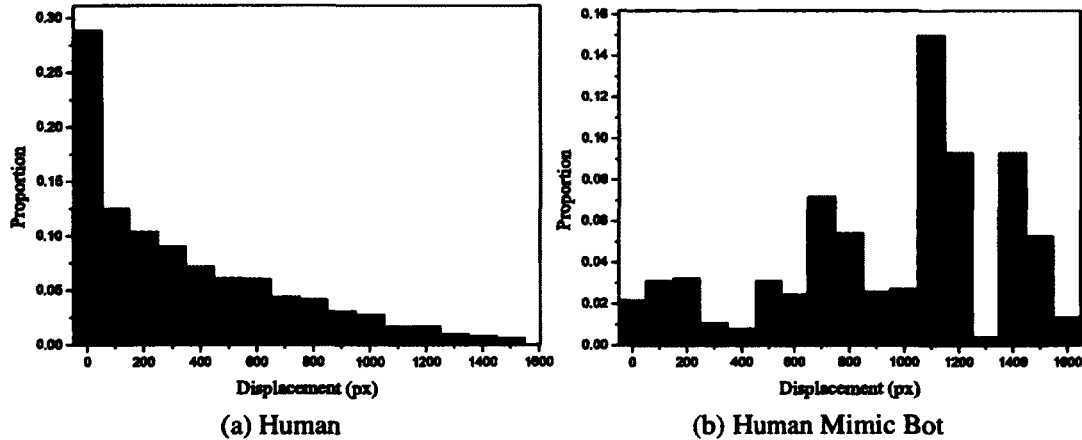


Figure 5.1: Displacement for Point-and-Click

constructs a syntactically legal HTTP response with the HTML form data as the body, and sends it to the submission URL at the server. To evade the server's check on the HTTP response, the bot often forges certain fields in the response header, such as Referer, User Agent, and Cookie. Furthermore, some bots are equipped with CAPTCHA deciphering capability to crack the CAPTCHA defense. However, they do not generate any mouse or keystroke events. Currently this type of bot is the most widely used blog bot in cyberspace [19, 58].

Contemporary detection methods have realized the importance of detecting human activities during the form filling procedure. A server only accepts a user as human if mouse or keyboard events are detected. Thus, bot authors are motivated to create a more advanced bot type, namely the Human Mimic Bot. These bots open a blog page in the browser, and use OS API calls to generate keystroke and mouse events. In this manner, it mimics human browsing behavior, fooling older detection methods. For example, the bot strolls down the page to the bottom by repetitively sending "Press down-key" commands. Then, it moves the mouse cursor into each field of the comment form, and types in prepared text content by sending a sequence of keystrokes. Finally, the bot posts the

---

fields and fill in appropriate content.



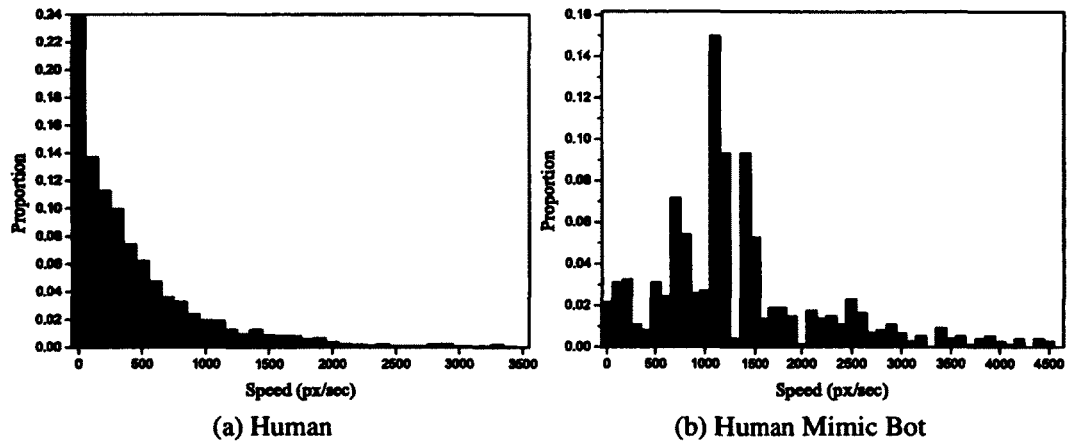


Figure 5.2: Speed for Point-and-Click

comment by generating a mouse click on the submit button. The server cannot distinguish whether the UI events are generated via hardware (such as the mouse device and keyboard) or via software (such as Human Mimic Bot) by merely checking the received user input data. The server will be deceived by Human Mimic Bot if it only relies on the presence of UI events for bot detection.

Some research into behavioral biometrics has found out that human behavior is more complex than bot behavior. Compared with the inherent irregularity and burstiness of human behavior, bots exhibit regular patterns of limited variety [88]. For example, many bots move the mouse cursor in straight lines at a constant speed, or strike keys with even intervals. Such perfect regular actions cannot be achieved by human. Thus, the server could detect Human Mimic Bot by taking behavioral complexity into account. With high fidelity of mimicry, Replay Bots are more advanced than Human Mimic Bots, and are probably the most difficult to detect among contemporary blog bots. When a human is filling a form, Replay Bot records her actions. Later on, it impersonates the human by replaying recorded traces on form submission pages. The standard interfaces utilized by popular blogs and message boards, such as WordPress or vBulletin, make such replay attacks possible.

To characterize the bot behaviors, we use existing bot tools or libraries to configure the three

types of blog bots. The Form Inject Bot is implemented as a PHP cURL script. The comment form at our blog site is submitted via the POST method. The cURL script assigns every input field with an appropriate value, encapsulates the form data into a string, and submits it to the PHP script at the server that processes the form. We configure the Human Mimic Bot based on the AutoHotkey script [12], which is an open-source Windows program designed for automating the Windows GUI and for general scripting<sup>6</sup>. We customize the script for our blog site, and thus it can generate actions corresponding to the page layout<sup>7</sup>. The script mimics all kinds of normal human actions, such as moving and clicking the mouse cursor, scrolling the page up and down, drag-and-dropping an area, and typing keys. To simulate various effects, we assign action parameters with different constants or random values. Taking mouse movement as an example, we change endpoint coordinates and movement speed to generate different traces. For keystrokes, we change the duration (the length of time the key is held) and inter-arrival time (the time from pressing one key to another) to generate different typing rhythms. We choose the Global Mouse and Keyboard Library for Windows [28] as the Replay Bot in our experiments, which has both record and replay capabilities. The record and replay are implemented using the mouse and keyboard APIs in Windows. Specifically, for recording, global hooks are created to capture keyboard and mouse events; and for replaying, the `keybd_event` and `mouse_event` APIs in Windows are used.

### 5.2.2 UI Data Collection

For client-side monitoring, we develop a logger written in JavaScript, which is embedded in the header template of every webpage, and in this way it records UI data during the user's entire visit at

---

<sup>6</sup>There are other similar bot tools that may generate simple human behavior, such as AutoIt [13] and AutoMe [14].

<sup>7</sup>The page layout is different from page to page, and may affect how the Human Mimic Bot works. For example, by moving down the same amount of pixels, the mouse enters the comment form on one page, but falls out of the form on another page.

the site. The user behavior is in constant monitoring, which prevents bots from bypassing routing checkpoints (such as CAPTCHA recognition during login). More specifically, five raw UI events generated by the user in the browser are collected, including Key Press, Key Release, Mouse Move, Mouse Button Press, and Mouse Button Release. The logger streams the UI data to the server for further processing and classification. More details of the logger implementation are presented in Section 5.3.1. Note that no user sensitive data content (e.g., password) is recorded by our logger. We have also obtained the approval from the Institutional Review Board (IRB) of our university, which ensures the appropriate and ethical use of human input data in our work.

The collection of human UI data is described as follows. We collected data from a busy blog site consisting of over 65,000 members. The site averages 800 simultaneous online users, and in order to prevent spam, the site requires visitors to register with real credentials and log in before posting content. Content is manually reviewed by site administrators, moderators, and a community of dedicated users. Should an account post spam and be reported, the associated content is quickly removed and the account gets suspended. We collected data from 1,078 distinct signed-in site members during several two-hour monitoring sessions on a single day. The data collection was completely transparent to users, and the interactions consist of both reading and posting of content. Our real-world data with the large user population covers a wide range of human input behavior. The data also presents an advantage over the lab environment tests, where a user's performance might be at odds with her normal behavior. We maintain a high degree of confidence that the users in this dataset are indeed human, as their registrations are manually screened by site administrators, and posted content is screened by a community of users, resulting in a low overall observed incidence of spam.

Correspondingly, we run three types of blog bots to collect bot input data. By including user-

name and password to the POST data body, Form Inject Bots can post comments. As it does not open a webpage in the browser to generate any input events, the server does not receive any UI data. Thus, Form Inject Bots can be easily detected. We also run multiple instances of the Human Mimic Bot, and each instance is assigned with different settings (such as varied typing rhythms and mouse movement speeds) to generate different behavior. We generate the traces of Human Mimic Bot for 30 hours. We run the Replay Bot for six rounds, which last for 2 hours in total. In each round, a human user fills in the comment form, and Replay Bot records the human trace and replays it.

Lastly, we explain the reasons that we run customized bots in the controlled “sand box” to generate bot input data. First, ground truth creation and data collection is an example of the chicken or the egg causality dilemma. We must know the true identity of a user to label it as human or bot in the ground truth set. In other words, we cannot collect data in the wild and recognize what data are generated by bot or not. After being trained on the ground truth set, the classifier can distinguish between human and bot. Second, we do not create bots. Instead, we customize bots based on existing tools and libraries without changing their mechanisms. The authenticity of bot input data is reserved. In addition, a bot needs to be customized to operate on a specific blog site<sup>8</sup>, and no existing tools can be generative to all blogs.

Raw UI events cannot efficiently describe user browsing activities. We develop a parser to integrate raw events into compound actions as shown in Table 5.1. For example, the Key Press event and the following Key Release event of the same key is integrated as a Keystroke action, and a set of continuous Mouse Move events are grouped as a Point action.

---

<sup>8</sup>For example, the position of the submit button may vary in the webpage layout. The bot must be customized to move to the button and generate a click event on it.

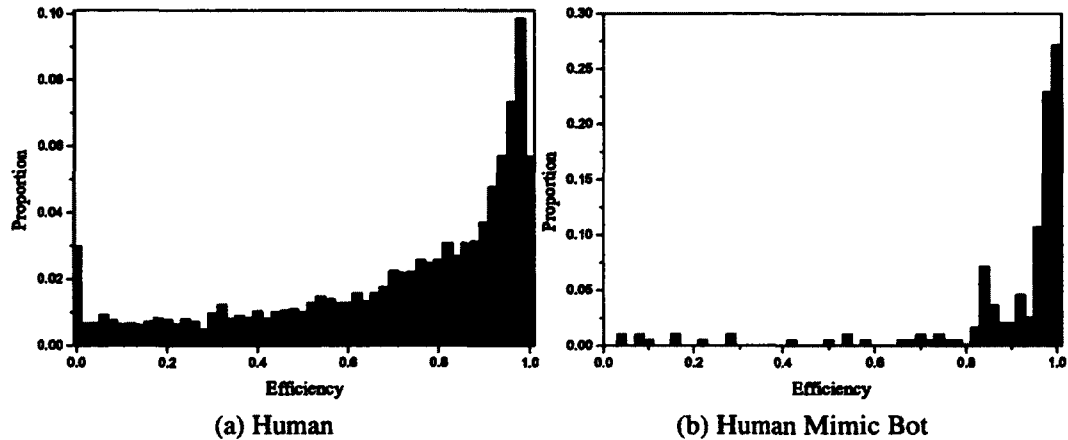


Figure 5.3: Movement Efficiency for Point-and-Click

### 5.2.3 UI Data Measurements

Based on the collected UI data from human and bot, we analyze the keystroke and mouse dynamics and characterize different behavioral patterns for humans and bots, respectively. For the profiling of bot behavior, we only use the traces of Human Mimic Bot, and exclude those of Form Inject Bot and Replay Bot<sup>9</sup>.

Figure 5.1 and 5.2 illustrate two mouse kinematics features, displacement and speed, for the Point-and-Click action, respectively. In Figure 5.1 with the bin resolution of 100 pixels, we observe that human users generate far more displacements with short length than with long length. About 60.64% of displacements are less than 400 pixels, while only 8.52% are greater than 1000 pixels. In contrast, bots tend to move the mouse at all displacements. Figure 5.2 with the bin resolution of 100 pixels per second shows the movement speed of bot is faster than that of human. The average speed of bot is 1520.83 pixels per second in our observation, but the average speed of human is 427.43 pixels per second. Furthermore, human speed is limited within 3500 pixels per second, due to the

<sup>9</sup>Form Inject Bot generates no UI data. As Replay Bot replays traces generated by human, it is inappropriate to include human traces to characterize bot behavior.

physical movement constraints of human wrist and arm. Finally, we observe that some bots move the mouse at fixed speeds.

Figure 5.3 shows the mouse movement efficiency for the Point-and-Click action, with the bin resolution of 0.02 second. For a mouse movement from the starting point to the end point, displacement is the segment length between the two points, and distance is the actual length traversed. Movement efficiency is defined as the ratio of displacement over distance. Straight line movement has the highest efficiency at 1. The more curvy the movement is, the lower its efficiency is. Our first observation is that bots move the mouse cursor with much greater efficiency than humans. About 59.23% of bot movements achieve efficiency greater than 0.94, while only 28.60% of human movements are equally efficient. As the Point action is the integration of a set of continuous raw Mouse Move events, we could have treated several segments of Move event as the curve of Point action, which lowers the bot efficiency during the calculation. Thus, there could have been more bot movements with the efficiency of 1 (namely, straight movement). Our second observation is that, the probability of human movement efficiency follows a lognormal (3P) distribution in our dataset<sup>10</sup>, and the bot probability does not fit any well-known distributions. For humans, most movements are curves, since it is physically difficult to generate perfect straight lines over certain length or time.

Figure 5.4 shows the distribution of inter-arrival times for the Keystroke action, with a bin resolution of 0.05 second. We make two observations from the figure. First, bots strike keys obviously faster than humans. About 21.49% of bot keystrokes are less than 0.05 second, and only 5.82% of human keystrokes are issued within that range. A human user has to look up keys on the keyboard, and moves her fingers to hit keys. Physical movements cannot compete with keystroke events generated by software. Second, for bots, the probabilities of intervals at 0.05 and 0.25 seconds are greatly

---

<sup>10</sup>Kolmogorov-Smirnov test presents P-value of the distribution fitting at 0.882 with a 99% confidence level.

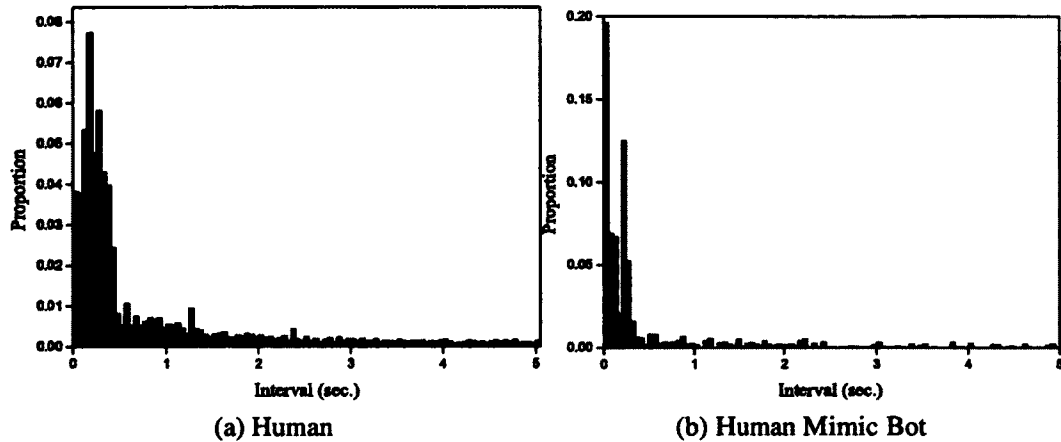


Figure 5.4: Inter-arrival Time Distribution for Keystroke

higher than other values. This implies that some bots may use periodic timers to issue keystrokes at fixed intervals.

We also observe similar distribution patterns of Keystroke duration between human and bot. The keystroke duration is the elapsed time between a key press and its corresponding release. The distribution patterns are similar with those in Figure 5.4. Bots hold keys much shorter than humans. While 45.42% of bot keystrokes are held less than 0.3 second, only 23.11% of human keystrokes are within that range. A human needs time to move his finger up to release the key after he presses it down. In addition, for bots, the probability of intervals between 0.05 and 0.15 seconds are greatly higher than other values. The periodic timer may set fixed intervals between consecutive key press and release events. Due to the space limit, the related figures are not included in the chapter.

### 5.3 System Design

Our detection system is mainly composed of the webpage-embedded logger and the server-side detector. The logger collects UI activities in the client browser and sends data to the server. The

detector analyzes the UI data of a user and decides whether it is human or bot. The high-level system architecture is shown in Figure 5.5.

### 5.3.1 Webpage-embedded Logger

As mentioned in Section 5.2.2, the logger is implemented as JavaScript code, and embedded in every webpage of the blog site. As a result, JavaScript is required by the blog site and non-JavaScript clients are blocked from posting or must pass a conventional HIP, such as a CAPTCHA. When a user visits the blog, the logger runs silently inside the client browser. It is totally transparent to the user, and no extensions need to be installed. The logger collects five raw UI events generated by the user inside the browser, including Key Press, Key Release, Mouse Move, Mouse Button Press, and Mouse Button Release. Each event is associated with a JavaScript listener. After an event happens, the listener is triggered to generate a record in the JSON format [36]. Every record has several fields to describe the event attributes<sup>11</sup>. The polling rate of the logger is decided by the client operating system, and is generally high enough to capture UI events. For example, in Windows 7, the polling rate is 125Hz, namely polling every 8 milliseconds. The logger buffers the collected events within a small time window, and then sends the data in a batch to the server via Ajax (Asynchronous JavaScript and XML). The asynchronous communication mechanism helps save network traffic between server and client, as no additional traffic occurs when no events happen within the window. Besides, according to Section 5.4.2, only a certain number of user actions are needed to correctly classify a user. It also helps reduce network traffic.

---

<sup>11</sup>Take the following Mouse Move record as an example, {"time":1278555037098, "type":"Mouse Move", "X":590, "Y":10, "tagName":"DIV", "tagID":"footnote"}. The "time" field contains the time stamp of the event in the unit of millisecond. The two coordinates, X and Y, denote the mouse cursor position. The last two fields describe the name and ID of the DOM element where the event happens, such as <div ID="footnote">. In a record of Mouse Press, {"time":1278555074750, "type":"Mouse Press", "virtualKey":0x01, "tagName":"HTML"}, The "virtualKey" field denotes the virtual-key code of 0x01 in hexadecimal value, which corresponds to the left mouse button here.



As our detection method is generic to other types of form bots, such as those automatically perform massive account registration and online voting, we need to address the privacy and security concerns of using the logger to collect user input data. First, we discuss the user privacy protection. As the logger is implemented as JavaScript code running in web pages of the blog site, it is strictly constrained by the same-origin policy [99] enforced by the browser, and thus cannot access content of other sites or programs. This makes it very different from the OS-level keyloggers. In other words, our logger can only access the data that a user generates on the blog, which will be submitted to the blog site anyway. Thus, the logger does not endanger user privacy. Second, we consider the confidentiality of user input content transferred over the Internet. When a user types in content on the webpage, the key values of strokes are recorded in the format of virtual-key codes [61]. The link between the logger and the server is not encrypted. To prevent an eavesdropper from intercepting data packages in plain text and recovering the user input content, the logger replaces each key value of strokes with a wildcard character. This wildcard replacement enforces the confidentiality of user input content, and avoids the additional overhead by encryption.

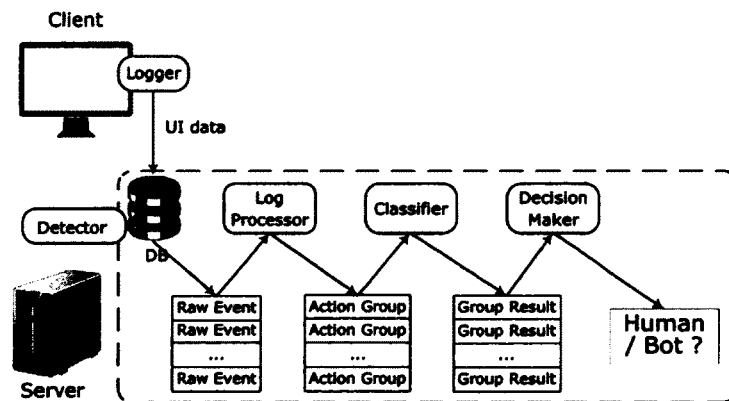


Figure 5.5: Detection System Architecture

### 5.3.2 Server-side Detector

The detector consists of three components: the log processor, the classifier, and the decision maker. The UI data of each user is processed by the log processor, which converts raw events into high-level actions and encapsulates an adjustable number of consecutive actions to form action groups. The classifier processes each action group in the user log and assigns it with a classification score, indicating how likely the action group is generated by human or bot. Finally, the decision maker determines the class of the user based on the classification results of action groups. Each of the components is explained as follows.

#### 5.3.2.1 Log Processor

When the UI data arrives at the server, it is in the format of raw events, such as Mouse Move and Key Press. The raw data is stored at the back-end MySQL database, and can be easily grouped per user who generates the data. Before classifying a user, the log processor processes the user log by converting raw events into high-level UI actions defined in Table 5.1. Furthermore, the log processor calculates the timing entropy of intervals of the whole raw event sequence in the user log, which detects periodic or regular timing of the entire user behavior.

The human behavior is often more complicated than that of bot [87, 78], which can be measured by entropy rate. It is a measure of the complexity of a process [79]. A high entropy rate indicates a random process, whereas a low indicates a regular process. The entropy rate is defined as the conditional entropy of an infinite sequence. As our real dataset is finite, the conditional entropy of finite sequences is used to estimate the entropy rate. For estimation, we use the corrected conditional entropy [122], which is defined as follows.

A random process  $X = \{X_i\}$  is defined as a sequence of random variables. The entropy of such

a sequence is defined as:

$$E(X_1, \dots, X_n) = - \sum_{i=1}^n P(x_1, \dots, x_n) \log P(x_1, \dots, x_n), \quad (5.1)$$

where  $P(x_1, \dots, x_n)$  is the joint probability  $P(X_1 = x_1, \dots, X_n = x_n)$ .

Thus, the conditional entropy of a random variable is:

$$E(X_n | X_1, \dots, X_{n-1}) = E(X_1, \dots, X_n) - E(X_1, \dots, X_{n-1}). \quad (5.2)$$

Then the entropy rate of a random process is defined as:

$$\bar{E}(X) = \lim_{n \rightarrow \infty} E(X_n | X_1, \dots, X_{n-1}). \quad (5.3)$$

The corrected conditional entropy is computed as a modification of Equation 5.3. First, the joint probabilities,  $P(X_1 = x_1, \dots, X_n = x_n)$  are replaced with empirically-derived probabilities. The data is binned into  $Q$  bins, i.e., values are converted to bin numbers from 1 to  $Q$ . The probabilities are then determined by the proportions of bin number sequences in the data. The entropy estimate and conditional entropy estimate, based on empirically-derived probabilities, are denoted as  $EN$  and  $CE$ , respectively. Second, a corrective term,  $perc(X_n) \cdot EN(X_1)$ , is added to adjust for the limited number of sequences for increasing values of  $n$  [122]. The corrected conditional entropy, denoted as  $CCE$ , is computed as:

$$\begin{aligned} CCE(X_n | X_1, \dots, X_{n-1}) = \\ CE(X_n | X_1, \dots, X_{n-1}) + perc(X_n) \cdot EN(X_1) \end{aligned} \quad (5.4)$$

Based on Equation 5.4, we calculate the  $CCE$  of intervals of the raw event sequence for a user as the timing entropy.

Finally, a set of classification features are generated for every action, which are listed in Table 5.2. They are used by the machine-learning based classifier for bot detection. More specifically, we group raw UI events into an action record as shown in Table 5.1. For example, a “Point” action contains a set of mouse move events. The value of duration feature is the timestamp difference between the last and first mouse move events. Similarly, the value of distance feature is the actual length traversed by all the mouse move events. The former seven features are directly retrieved from the action itself. In particular, the first four features are the basic ones, while average speed and move efficiency are derived from them<sup>12</sup>. These two derived features reveal the inherent correlation among features and accelerate the tree building. The last feature is the timing entropy of the whole event interval sequence of a user, not of a single action. An action only consists of several events, which are too few to extract timing regularity. It is statistically meaningful to calculate entropy at the user level. We include the entropy feature in the action record to inform the classifier the behavioral timing pattern of the user who generates the action.

**Table 5.2: Classification Features of User Actions**

Feature	Description
Duration	Mouse/keystroke actions
Distance	Mouse actions
Displacement	Mouse actions
Displacement Angle	Mouse actions
Average Speed	Mouse actions
Move Efficiency	Mouse actions
Virtual Key Value	Left/middle/right button for mouse actions, and a wildcard character for keystrokes
Timing Entropy	Event interval sequence of the target user

<sup>12</sup>Average speed is distance over duration, and move efficiency is displacement over distance.

### 5.3.2.2 Classifier

Our classifier is based on the C4.5 algorithm [105] that builds a decision tree for classification. The decision tree predicts the class of an unknown sample based on the observed attributes. There are two types of nodes in the decision tree, the leaf node labeled with the class value (such as human or bot), and the interior node that corresponds to an attribute and links to a subtree. The tree is constructed by dividing the training dataset into subsets based on the attribute value test. This partitioning process is executed on each derived subset in a recursive manner. The fundamental ideas behind C4.5 are briefly described as follows. The tree is built from the root downward to leaves. During the construction path, each interior node must be associated with the attribute that is most informative among the attributes not yet included in the path. C4.5 uses entropy to measure how informative an attribute is. Given a probability distribution  $P = \{p_1, p_2, \dots, p_n\}$ , the entropy of  $P$  is defined as

$$E(P) = - \sum_{i=1}^n p_i \log p_i, \quad (5.5)$$

We denote  $D$  as the dataset of labeled samples, and  $C$  as the class with  $k$  values,  $C = \{C_1, C_2, \dots, C_k\}$ .

The information required to identify the class of a sample in  $D$  is denoted as  $Info(D) = E(P)$ , where  $P$ , as the probability distribution of  $C$ , is

$$P = \left\{ \frac{|C_1|}{|D|}, \frac{|C_2|}{|D|}, \dots, \frac{|C_k|}{|D|} \right\}. \quad (5.6)$$

If we partition  $D$  based on the value of an attribute  $A$  into subsets  $\{D_1, D_2, \dots, D_m\}$ ,

$$Info(A, D) = \sum_{i=1}^m \frac{|D_i|}{|D|} Info(D_i). \quad (5.7)$$

After the value of attribute  $A$  is obtained, the corresponding gain in information due to  $A$  is denoted

as

$$Gain(A, D) = Info(D) - Info(A, D), \quad (5.8)$$

As *Gain* favors attributes that have a large number of values, to compensate for this the C4.5 algorithm uses *Gain Ratio* as

$$GainRatio(A, D) = \frac{Gain(A, D)}{SplitInfo(A, D)} \quad (5.9)$$

where  $SplitInfo(A, D)$  is the information due to the splitting of  $D$  based on the value of attribute  $A$ .

Thus,

$$SplitInfo(A, D) = E\left(\frac{|D_1|}{|D|}, \frac{|D_2|}{|D|}, \dots, \frac{|D_n|}{|D|}\right) \quad (5.10)$$

The gain ratio is used to rank how informative attributes are and to construct the decision tree, where each node is associated with an attribute having the greatest gain ratio among the attributes not yet included in the path from the root. In other words, C4.5 applies a greedy search by selecting the candidate test that maximizes the heuristic splitting criterion.

We choose the C4.5 algorithm for the classification due to the following four reasons. First, it builds the decision tree in an efficient manner by processing a large amount of training data in a short time. Furthermore, the tree is robust even if assumptions, to some extent, are violated by the real data model. Second, it uses the white box model, which is easy to understand and interpret by boolean logic. Third, C4.5 is capable of processing both continuous and discrete values (such as numerical and categorical data), which is an improvement from the earlier ID3 algorithm [124]. Last, after the tree creation, C4.5 prunes the tree from top down with attempts to constrain the tree height and avoid overfitting.

We use J48 as implementation, which is an open source Java program of the C4.5 algorithm in the Weka data mining tool [93]. Each action record is in such a format of feature vector as

*<duration, distance, displacement, displacement angle, average speed, move efficiency, virtual key value, timing entropy>*, listed in Table 5.2. The J48 classifier takes input from all actions in an action group<sup>13</sup>, and outputs the classification result indicating whether the action group is generated by human or bot.

### 5.3.2.3 Decision Maker

The user log contains multiple action groups, and each group is determined by the classifier as generated by either human or bot. The decision maker presents the summary of the classifications of UI actions over a period of time by employing the majority voting rule. More specifically, if the majority<sup>14</sup> of action groups are classified as human, then the user is classified as human, and vice versa. Since classification on individual actions cannot always be accurate, the more actions are included, the more confident the final decision is.

## 5.4 Evaluation

In this section we evaluate the efficacy of our detection system in terms of detection accuracy, detection time, and induced system overhead.

### 5.4.1 Experimental Setup

Our experiments are based on 239 hours of user traces, including 207 hours of human and 32 hours of bot<sup>15</sup>. The traces are collected from more than 1,000 human users and two types of blog bots

---

<sup>13</sup>Input is converted the ARFF format required by Weka[11].

<sup>14</sup>As our classification only involves two categories, human and bot, a majority means more than half of the votes.

<sup>15</sup>The idle time is not included in the traces. The bot trace consists of 30 hours of Human Mimic Bot data and 2 hours of Replay Bot data.

(namely Human Mimic Bot and Replay Bot). The details about user composition are described in Section 5.2.2. In summary, the user input dataset consists of 4,520,165 raw events, which are further converted into 190,677 compound actions.

**Table 5.3:** True Positive and Negative Rates vs No. of Actions per Group

Actions per Group	TPR	TNR
2	0.974	0.9993
4	0.9945	0.9996
6	0.9865	0.9989
8	0.9879	0.9989

We use *cross validation* with ten folds [111] to train and test the classifier on our UI dataset. The dataset is randomly partitioned into ten complementary subsets. In each round, one of the ten subsets is retained to validate the classifier (as the test set), while the remaining nine subsets are used to train the classifier (as the training set). Every round is an independent procedure, as the classifier is reset at the beginning of the round and then re-trained. The test results from ten rounds are averaged to generate the final estimation. The advantage of cross validation is that, all the samples in the dataset are used for both training and validation and each sample is validated exactly once.

#### 5.4.2 System Performance

Our detection system has two adjustable parameters that affect the system performance: (1) the number of actions per group and (2) the total number of actions required to correctly classify a user. We describe the configuration procedure of each parameter as follows.

We set different values for the number of actions per group, run cross validation tests, and then



calculate the true positive rate (TPR)<sup>16</sup> and true negative rate (TNR)<sup>17</sup> for each value. The results are listed in Table 5.3. During the classification, the classifier treats a group of actions as one entity<sup>18</sup>, and produces the classification result for the group, not for individual actions. In our experiment, the setting of four generates the highest TPR and TNR among all the values. Therefore, we set the number of actions per group as four.

The second parameter, the total number of actions required to correctly classify a user, directly affects the system performance in terms of detection accuracy and detection time. Generally speaking, the more actions observed from the user, the more accurate the classification result will be. On the other hand, processing more actions costs more time and increases the detection time. Given the number of actions per group is four, we run experiments with cross validation on the whole ground truth to determine how many actions are required to achieve a high accuracy. The results are summarized in the column labeled as “Both Bots” in Table 5.4. Since each action group is configured to contain four actions, the total number of actions required equals the group number multiplied by four. The last row in Table 5.4 labeled as “Entire” corresponds to the baseline case, in which the classifier takes all the actions in the user log as input. It is used as upper-limit for accuracy comparison. We can see that the detection accuracy in terms of TPR and TNR increases as the total number of actions processed by the classifier increases. With the group number as 24 (namely  $24 * 4 = 96$  actions in total), TPR and TNR are very close to those of the entire log. Besides, the accuracy gain increases very slowly after the group number exceeds 24. Thus, the system is configured to process 24 action groups while each group includes 4 actions. Each group is labeled as either human or bot,

---

<sup>16</sup>The true positive rate is the ratio of the number of bots which are correctly classified to the number of all the bots.

<sup>17</sup>The true negative rate is the ratio of the number of humans which are correctly classified to the number of all the humans.

<sup>18</sup>A series of consecutive actions represent continuous behavior well.

and the user is eventually classified as the category with more labels using the majority voting rule. For example, if the action group sequence is labeled as  $\langle \text{human, human, bot, human, } \dots, \text{human} \rangle$ , then the user is classified as human. The C4.5 algorithm generates a decision tree based on our dataset and prunes it afterwards. The construction procedure costs 4.96 seconds, and returns a tree with 57 nodes. The tree consists of 29 leaves and 28 interior nodes including the root. The overall detection accuracy is 0.9972 with the root mean squared error at 0.0244.

**Table 5.4:** True Positive and Negative Rates vs Number of Groups

Group No	Both Bots		Human Mimic Bot		Replay Bot	
	TPR	TNR	TPR	TNR	TPR	TNR
4	0.6975	0.9972	0.7016	0.998	0.6359	0.9992
8	0.7673	0.9956	0.7710	0.9982	0.7117	0.9974
12	0.8172	0.9973	0.8198	0.9991	0.7781	0.9982
16	0.8788	0.9978	0.8802	0.9992	0.8578	0.9986
20	0.917	0.9982	0.9208	0.9994	0.8599	0.9988
24	0.9794	0.9983	0.9817	0.9996	0.9448	0.9987
Entire	0.9945	0.9996	0.9964	0.9999	0.9660	0.9997

The detection time is mainly decided by the total number of actions processed by the classifier. The average time per action is less than one millisecond. The overall time cost per user, including log processing and classification, is averagely 3.2 seconds.

We speculate whether one bot type is more difficult to detect than the other. Thus, we separate the evaluation on Human Mimic Bot and Replay Bot to see how accurately our system can detect the two types of blog bots. More specifically, we derive two subsets of the ground truth: one with the entire trace of human and Human Mimic Bot, and the other with that of human and Replay Bot. The results are displayed in the last two columns in Table 5.4. We have two observations. Firstly, for each row, the TPR of Human Mimic Bot is greater than that of Replay Bot. It is easier to detect Human Mimic Bot thanks to the simplicity and regularity of its behavior. Due to certain

implementation deficiencies of the Replay Bot tools<sup>19</sup>, our system also effectively detects Replay Bot with the TPR greater than 0.966. Secondly, the TNR is greater than the corresponding TPR for every bot type. In other words, the FNR is greater than the FPR. It reflects our design philosophy that, the system may miss capturing some bots, but it seldom mis-classifies human as bot to upset legitimate users.

### 5.4.3 System Overhead

As the detector is employed on the server side, it must be light-weight and scalable enough to accommodate numerous concurrent user classifications. We estimate the additional overhead induced by the detector for the case, in which 10,000 users access the server simultaneously.

In terms of network bandwidth consumption, the logger streams the user input data in the JSON format to the server. An average user generates a trace at a size around 200 Kbytes. Then, the aggregated network bandwidth consumed at the server-side for receiving UI data is about 4.2 Mbps. Considering the wide deployment of Gigabit Ethernet, this network bandwidth requirement can be easily met.

The main memory cost at the server side is to accommodate user input actions and the decision tree outputs for each user. An input action contains eight features, and each feature occupies 5 bytes, except the virtual key value with 2 bytes. Thus, a single action consumes 37 bytes. Each action group contains 4 actions, and is assigned with a result that occupies 1 byte. The detector only needs 24 action groups from the user log for classification, and thus classifying a single user consumes up to 3.49 Kbytes of memory. Scaled to 10,000 online users, the memory cost of the server will be 34.1 Mbytes, which is very affordable for a modern server.

---

<sup>19</sup>More explanations are made in Section 5.5.

The computational overhead is also very minor. We run J48 in the Weka, a Java implementation of the C4.5 algorithm, on a workstation with an Inter Core 2 Duo 2.4 GHz CPU. The classification time is 10.85 seconds for the traces of 239 hours.

## 5.5 Discussion

Once attackers know the existence of our detection system, they will attempt to evade it. An adversary could directly send synthetic traces to the server, trying to deceive the detector. We discuss the trace forging in three aspects. First, the adversary records the trace when a human fills in the form, and replays it. Some current record-and-replay tools, such as the Global Mouse and Keyboard Library for Windows [28], cannot restore human trace with high fidelity and thus create detectable artifacts. For example, the timing of events is more regular in the replayed trace than the original human trace, which can be detected based on its regular and low entropy pattern. The difference can be identified by our detector.

Second, suppose the bot tool would perfectly replay human trace. Then, the adversary has to record a different trace in real-time (typing them out by a person) for each different spam message it wants to post. Therefore, our detection system will at least significantly raise the bar against blog bots and their spamming cost. Third, as an alternative of replay, the adversary might be motivated to develop a complete generative model of human user-input dynamics, and send the bogus trace to the server bypassing the client-side logger. However, the inherent complexity and uncertainty of human behavior makes it a very difficult modeling problem, because no such a model exists yet according to our best knowledge.

Now we discuss the limitation that enabling JavaScript in a browser is required for the blog sites

protected by our system. With many popular sites, such as Twitter, Facebook, and Digger, being entirely dependent on JavaScript for key tasks [33] such as posting, we do not feel this requirement will cause any problem for most human users. According to a study of Internet user browser settings [47], only a tiny proportion (around 0.56%) of human users turn off JavaScript in practice. In other words, most human users will successfully generate UI data to the logger and pass our behavior detection even without noticing it. Bots could disable JavaScript to prevent the logger from obtaining their genuine traces. However, the disabled JavaScript will reveal the high likelihood of a bot being behind the browser. Meanwhile, to cope with this situation, a server can take one step back and resume the use of the conventional HIP methods, such as CAPTCHA, to defend against bots.

Lastly, we discuss the feasibility of applying our work to other interactive web applications. The core idea of this chapter is detecting human participation by analyzing the user input data, and it can be applied to those web applications that require human participation, such as forums and online social networks. However, developers need to fulfill some tasks during transplantation. They need to collect the ground truth data for their own applications, as user behavior (both of human and bot) may be application specific. Some features may also have to be replaced for accurate classification.

## 5.6 Conclusion

This chapter presents a blog bot detection system, which leverages the behavioral differences between human users and bots in their mouse and keystroke activities. Compared to conventional detection methods based on Human Interactive Proofs, such as CAPTCHA, our detection system does not require additional user participation, and is thus both transparent and unobtrusive to users. We have collected real user input traces of 239 hours from a busy blog site. Based on these real

UI traces, we have discovered different user behavioral characteristics, and further developed useful features for classification. Our detection system consists of a webpage-embedded logger and server-side detector. The logger passively collects user activities and streams this data to the server. The detector processes the log and identifies whether it is generated by human or bot. The core of our detection system is the C4.5 algorithm that builds a decision tree. It takes the action stream as input, and classifies the user by the majority voting rule.

We perform a set of experiments to tune the system parameters and evaluate the system's performance. The experimental results show that the overall detection accuracy is over 99%. The additional overhead induced by the detection is minor in terms of CPU and memory costs. Since the detection only requires user input traces, our method can be applied to detecting other types of form bots, such as account registration and online voting.

# Chapter 6

In the previous chapters, we have detailed four problems and their solutions in the research area of detecting abnormal behaviors in Web applications. This chapter concludes the dissertation and outlines the future work.

## 6.1 Conclusions

Web applications have become a vital part of our daily lives. Unsurprisingly, network attacks have exploited various vulnerabilities of web applications, and caused substantial damages to Internet users. Anomaly detection is an important area of web security, which is especially effective in detecting unknown attacks. This dissertation focuses on detecting abnormal behaviors in web applications by employing the following methodology. For a web application, we have conducted a set of measurements to reveal the existence of abnormal behaviors in it. Then, we have observed the differences between normal and abnormal behaviors. By applying a variety of methods in information extraction, such as heuristics algorithms, machine learning, and information theory, we have extracted features useful for building a classification system for detecting abnormal behaviors.

Specifically, we have investigated four challenging problems of anomaly detection: web resource hotlinking, aggressive automation of Twitter accounts, social spam campaigns on Twitter, and blog bots. We have proposed new solutions to address these problems, and developed system prototypes to validate the efficacy of our solutions. Our contributions include an anti-hotlinking

framework for protecting web resources on hosting servers, a classification system that detects and classifies automation degree of Twitter accounts, a classification system that clusters social campaigns on Twitter and differentiates spam campaigns from legitimate ones, and a blog protection framework that monitors user input based on behavioral biometrics and distinguishes between human bloggers and blog bots. Our experimental results have demonstrated that our proposed solutions are effective for tackling these real-world problems.

## 6.2 Future Work

We plan to continue the future work in the following directions. First, we will extend our methodology of anomaly detection to other interactive web applications, especially social networks. User interaction is the distinctive characteristic of social networks. Adversaries including attackers and spammers may launch a variety of unknown attacks by interacting with benign users. Although attack forms may vary, an effective defense is still to detect abnormal behaviors that deviate from normal behaviors.

Second, we plan to enhance web security in the environment of cloud computing. A feasible proposal is to implement the online detection of spam campaigns on Twitter or other social networking platforms. We have implemented the offline detection of spam campaigns on Twitter in Section 4. A big challenge for online detection is scalability and latency. Facing the huge user population and unpredictable traffic, the underlying cloud infrastructure may meet scalability and latency demands by dynamically allocating and releasing computing resources.

Third, we are very interested in how more advanced bots can mimic human behavior, and will pay close attention to the related research. In Section 5, we have used metrics including user input



dynamics and entropy to differentiate bot from human. Aware of our research, adversaries may be motivated to develop more advanced bots to evade the detection. For example, a bot may follow a verisimilar model of human user-input dynamics to generate input actions. To cope with that, we may need to develop new metrics to characterize the subtle behavioral difference between human and bot.

# Bibliography

- [1] 2mdn. <http://2mdn.net/> [Accessed: Nov. 5, 2011].
- [2] Ajax, asynchronous javascript and xml. <https://developer.mozilla.org/en/AJAX> [Accessed: Nov. 5, 2011].
- [3] Akamai, web application acceleration and performance management. <http://www.akamai.com/> [Accessed: Jan. 12, 2011].
- [4] Akismet, comment spam prevention for your blog. <http://akismet.com/> [Accessed: Aug. 23, 2011].
- [5] Alexa. <http://www.alexa.com/> [Accessed: Jan. 15, 2010].
- [6] Amazon comes to twitter. [http://www.readwriteweb.com/archives/amazon\\_comes\\_to\\_twitter.php](http://www.readwriteweb.com/archives/amazon_comes_to_twitter.php) [Accessed: Dec. 20, 2009].
- [7] The apache http server project. <http://httpd.apache.org/> [Accessed: Dec. 20, 2009].
- [8] The apache module mod\_bandwidth. [http://www.cohprog.com/mod\\_bandwidth.html](http://www.cohprog.com/mod_bandwidth.html) [Accessed: Dec. 20, 2009].
- [9] The apache module mod\_limitipconn.c. <http://dominia.org/djao/limitipconn.html> [Accessed: Dec. 3, 2009].
- [10] The apache module mod\_rewrite url rewriting engine. [http://httpd.apache.org/docs/1.3/mod/mod\\_rewrite.html](http://httpd.apache.org/docs/1.3/mod/mod_rewrite.html) [Accessed: Dec. 20, 2009].
- [11] Attribute-relation file format (arff). <http://www.cs.waikato.ac.nz/ml/weka/arff.html> [Accessed: Aug. 12, 2010].
- [12] Autohotkey - free mouse and keyboard macro program with hotkeys. <http://www.autohotkey.com/> [Accessed: Sept. 15, 2010].
- [13] Autoit, automation and scripting language. <http://www.autoitscript.com/site/autoit/> [Accessed: Mar. 17, 2011].
- [14] Autome - automate mouse and keyboard actions. <http://www.asoftech.com/autome/> [Accessed: Mar. 17, 2011].

- [15] Barack obama uses twitter in 2008 presidential campaign. <http://twitter.com/BarackObama/> [Accessed: Dec. 20, 2009].
- [16] The best free software of 2009, features by pc magazine. <http://www.pcmag.com/article2/0,2817,2338803,00.asp> [Accessed: Mar. 25, 2009].
- [17] Bestbuy goes all twitter crazy with @twelforce. [http://twitter.com/in\\_social\\_media/status/2756927865](http://twitter.com/in_social_media/status/2756927865) [Accessed: Dec. 20, 2009].
- [18] Blog top sites ranking. <http://www.blogtopsites.com/> [Accessed: Jun. 20, 2009].
- [19] Blogbot by incansoft. <http://blogbot.auto-submitters.com/> [Accessed: Oct. 9, 2010].
- [20] Blogflux top blog sites overall statistics. <http://topsites.blogflux.com/stats.php> [Accessed: Jun. 20, 2009].
- [21] Cachefly, content delivering. <http://www.cachefly.com/> [Accessed: Jun. 4, 2011].
- [22] Captcha: Telling humans and computers apart automatically. <http://www.captcha.net/> [Accessed: Jun. 4, 2011].
- [23] The digital millennium copyright act of 1998. <http://www.copyright.gov/legislation/dmca.pdf> [Accessed: Jun. 4, 2011].
- [24] Doubleclick. <http://www.doubleclick.com/> [Accessed: Jun. 4, 2011].
- [25] Flickr. <http://www.flickr.com/> [Accessed: Jun. 4, 2011].
- [26] Free software downloads and reviews by download.com. <http://download.cnet.com/Best-Free-Software/1200-20-5154518.html> [Accessed: Jun. 4, 2011].
- [27] Gd graphics library. <http://www.boutell.com/gd/> [Accessed: Jun. 4, 2011].
- [28] Global mouse and keyboard library. <http://www.codeproject.com/KB/system/globalmousekeyboardlib.aspx> [Accessed: Nov. 24, 2010].
- [29] Gnu wget. <http://www.gnu.org/software/wget/> [Accessed: Jun. 4, 2011].
- [30] Google adsense. <http://www.google.com/adsense/> [Accessed: Jun. 4, 2011].
- [31] Google safe browsing API. <http://code.google.com/apis/safebrowsing/> [Accessed: Aug. 27, 2011].
- [32] Google syndication. <http://googlesyndication.com/> [Accessed: Oct. 12, 2009].
- [33] How much of the web actually work without javascript. [http://tobyho.com/How\\_Much\\_of\\_the\\_Web\\_Actually\\_Work\\_Withou\\_Javascript](http://tobyho.com/How_Much_of_the_Web_Actually_Work_Withou_Javascript) [Accessed: Apr. 7, 2011].
- [34] Http state management mechanism, rfc2109. <http://www.ietf.org/rfc/rfc2109.txt> [Accessed: Oct. 12, 2009].

- [35] The hypertext transfer protocol (http), rfc2616. <http://www.w3.org/Protocols/rfc2616/rfc2616.html> [Accessed: Oct. 12, 2009].
- [36] Json, javascript object notation. <http://www.json.org/> [Accessed: Nov. 5, 2010].
- [37] Koobface. <http://en.wikipedia.org/wiki/Koobface> [Accessed: Aug. 17, 2011].
- [38] Megaupload: the leading online storage and file delivery service. <http://www.megaupload.com/> [Accessed: Oct. 12, 2009].
- [39] Phishtank, join the fight against phishing. <http://www.phishtank.com/> [Accessed: Aug. 27, 2011].
- [40] Php, hypertext preprocessor. <http://www.php.net/> [Accessed: Oct. 12, 2009].
- [41] Prototype javascript framework. <https://www.prototypejs.org/> [Accessed: Oct. 12, 2009].
- [42] Rapidshare: 1-click web hosting, easy filehosting. <http://www.rapidshare.com/> [Accessed: Oct. 12, 2009].
- [43] Senseclusters. <http://senseclusters.sourceforge.net/> [Accessed: Sept. 2, 2011].
- [44] Sophos security threat report: 2010. <http://www.sophos.com/sophos/docs/eng/papers/sophos-security-threat-report-jan-2010-wpna.pdf> [Accessed: Oct. 17, 2010].
- [45] The spamhaus project. <http://www.spamhaus.org/> [Accessed: Aug. 27, 2011].
- [46] State of twitter spam. <http://blog.twitter.com/2010/03/state-of-twitter-spam.html> [Accessed: Aug. 17, 2011].
- [47] A study of internet users' cookie and javascript settings. <http://smorgasbork.com/component/content/article/84-a-study-of-internet-users-cookie-and-javascript-settings> [Accessed: Apr. 7, 2011].
- [48] Surbl. <http://www.surbl.org/lists> [Accessed: Aug. 27, 2011].
- [49] tdash's api of twitter applications statistics. <http://tdash.org/stats/clients> [Accessed: Sept. 6, 2011].
- [50] The top 500 sites on the web by alexa. <http://www.alexa.com/topsites> [Accessed: Jan. 15, 2010].
- [51] Top trending twitter topics for 2011 from what the trend. <http://blog.hootsuite.com/top-twitter-trends-2011/> [Accessed: Dec. 15, 2011].
- [52] tweetadder, professional twitter marketing tool. <http://tweetadder.com/> [Accessed: Sept. 5, 2011].

- [53] Twitter blog: Avoid phishing scams. <http://blog.twitter.com/2010/02/avoid-phishing-scams.html> [Accessed: Aug. 17, 2011].
- [54] Twitter blog: Your world, more connected. <http://blog.twitter.com/2011/08/your-world-more-connected.html> [Accessed: Aug. 17, 2011].
- [55] Twitter rest api resources. <https://dev.twitter.com/docs/api> [Accessed: Aug. 30, 2011].
- [56] The twitter rules. <http://support.twitter.com/entries/18311-the-twitter-rules> [Accessed: Aug. 17, 2011].
- [57] Twitter's streaming api documentation. <https://dev.twitter.com/docs/streaming-api> [Accessed: Aug. 30, 2011].
- [58] Ultimate wordpress comment submitter. <http://www.wordpresscommentsspammer.com/> [Accessed: Oct. 9, 2010].
- [59] Uribl, realtime uri blacklist. <http://http://www.uribl.com/about.shtml> [Accessed: Aug. 27, 2011].
- [60] Using the twitter search api. <https://dev.twitter.com/docs/using-search> [Accessed: Aug. 30, 2011].
- [61] Virtual-key codes. <http://msdn.microsoft.com/en-us/library/ms927178.aspx> [Accessed: Nov. 5, 2010].
- [62] Yahoo! advertising. <http://advertising.yahoo.com/> [Accessed: Feb. 4, 2010].
- [63] BEN ADIDA. Sessionlock: securing web sessions against eavesdropping. In *Proceeding of the 17th international conference on World Wide Web*, pages 517–524, 2008.
- [64] AHMED AWAD E. AHMED AND ISSA TRAORE. A new biometric technology based on mouse dynamics. *IEEE Trans. Dependable Secur. Comput.*, 4:165–179, July 2007.
- [65] AKIKO AIZAWA. The feature quantity: an information theoretic perspective of tfidf-like measures. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 104–111, 2000.
- [66] LUCAS BALLARD, FABIAN MONROSE, AND DANIEL LOPRESTI. Biometric authentication revisited: understanding the impact of wolves in sheep's clothing. In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, 2006.
- [67] FABRICIO BENEVENUTO, GABRIEL MAGNO, TIAGO RODRIGUES, AND VIRGILIO ALMEIDA. Detecting spammers on twitter. In *Proceedings of the CEAS 2010*.
- [68] FRANCESCO BERGADANO, DANIELE GUNETTI, AND CLAUDIA PICARDI. User authentication through keystroke dynamics. *ACM Trans. Inf. Syst. Secur.*, 5:367–397, November 2002.
- [69] HAL BERGHEL. Hijacking the web. *Commun. ACM*, 45(4):23–27, 2002.

- [70] LEO BREIMAN. Random forests. *Machine Learning*, 45:5–32, 2001.
- [71] MARCUS BROWN AND SAMUEL JOE ROGERS. User identification via keystroke characteristics of typed names using neural networks. *Int. J. Man-Mach. Stud.*, 39:999–1014, December 1993.
- [72] MEEYOUNG CHA, HAEOON KWAK, PABLO RODRIGUEZ, YONG-YEOL AHN, AND SUE MOON. I tube, you tube, everybody tubes: analyzing the world’s largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, San Diego, CA, USA, 2007.
- [73] MEEYOUNG CHA, ALAN MISLOVE, AND KRISHNA P. GUMMADI. A measurement-driven analysis of information propagation in the flickr social network. In *Proceedings of the 18th International Conference on World Wide Web*, Madrid, Spain, 2009.
- [74] VARUN CHANDOLA, ARINDAM BANERJEE, AND VIPIN KUMAR. Anomaly detection : A survey. *ACM Computing Surveys*, 41(3), July 2009.
- [75] KUMAR CHELLAPILLA, KEVIN LARSON, PATRICE SIMARD, AND MARY CZERWINSKI. Designing human friendly human interaction proofs (hips). In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2005.
- [76] THOMAS CHEN AND PETER HENRY. Phishing and countermeasures: Understanding the increasing problem of electronic identity theft. *Journal of Digital Forensic Practice*, 1:147–149, July 2006.
- [77] N. CHOU, R. LEDESMA, Y. TERAGUCHI, D. BONEH, AND J. C. MITCHELL. Client-side defense against web-based identity theft. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium*, Alexandria, VA, USA, 2004.
- [78] ZI CHU, STEVEN GIANVECCHIO, HAINING WANG, AND SUSHIL JAJODIA. Who is tweeting on twitter: human, bot or cyborg? In *Proceedings of the 2010 Annual Computer Security Applications Conference*, Austin, TX, USA, 2010.
- [79] THOMAS M. COVER AND JOY A. THOMAS. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 2006.
- [80] JAMES DEMMEL AND WILLIAM KAHAN. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. and Stat. Comput*, 11:873–912, 1990.
- [81] MARCEL DISCHINGER, ANDREAS HAEBERLEN, KRISHNA P. GUMMADI, AND STEFAN SAROIU. Characterizing residential broadband networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet Measurement*, San Diego, CA, USA, 2007.
- [82] IL-CHUL MOON DONGWOO KIM, YOHAN JO AND ALICE OH. Analysis of twitter lists as a potential source for discovering latent characteristics of users. In *To appear on CHI 2010 Workshop on Microblogging: What and How Can We Learn From It?*, 2010.
- [83] GEORGES DUPRET. Latent concepts and the number orthogonal factors in latent semantic analysis. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 221–226, 2003.

- [84] HENRY J. FOWLER AND WILL E. LELAND. Local area network traffic characteristics, with implications for broadband network congestion management. *IEEE Journal of Selected Areas in Communications*, 9(7), 1991.
- [85] HONGYU GAO, JUN HU, CHRISTO WILSON, ZHICHUN LI, YAN CHEN, AND BEN Y. ZHAO. Detecting and characterizing social spam campaigns. In *Proceedings of the 10th annual conference on Internet measurement*, pages 35–47, 2010.
- [86] SAPTARSHI GHOSH, GAUTAM KORLAM, AND NILOY GANGULY. Spammers' networks within online social networks: a case-study on twitter. In *Proceedings of the 20th international conference companion on World wide web*, pages 41–42, 2011.
- [87] STEVEN GIANVECCHIO AND HAINING WANG. Detecting covert timing channels: An entropy-based approach. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security*, Alexandria, VA, USA, October–November 2007.
- [88] STEVEN GIANVECCHIO, ZHENYU WU, MENGJUN XIE, AND HAINING WANG. Battle of botcraft: fighting bots in online games with human observational proofs. In *Proceedings of the 16th ACM conference on Computer and Communications Security*, Chicago, IL, USA, 2009.
- [89] STEVEN GIANVECCHIO, MENGJUN XIE, ZHENYU WU, AND HAINING WANG. Measurement and classification of humans and bots in internet chat. In *Proceedings of the 17th USENIX Security symposium*, San Jose, CA, 2008.
- [90] MINAS GJOKA, MACIEJ KURANT, CARTER T BUTTS, AND ATHINA MARKOPOULOU. Walking in facebook: A case study of unbiased sampling of osns. In *Proceedings of the 27th IEEE International Conference on Computer Communications*, San Diego, CA, USA, March 2010.
- [91] PAUL GRAHAM. A plan for spam. <http://www.paulgraham.com/spam.html> [Accessed: Jan. 25, 2008].
- [92] CHRIS GRIER, KURT THOMAS, VERN PAXSON, AND MICHAEL ZHANG. @spam: the underground on 140 characters or less. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 27–37, 2010.
- [93] MARK HALL, EIBE FRANK, GEOFFREY HOLMES, BERNHARD PFAHRINGER, PETER REUTEMANN, AND IAN H. WITTEN. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, 2009.
- [94] MONIKA R. HENZINGER, ALLAN HEYDON, MICHAEL MITZENMACHER, AND MARC NAJORK. On near-uniform url sampling. In *Proceedings of the 9th International World Wide Web Conference on Computer Networks*, Amsterdam, The Netherlands, May 2000.
- [95] TIN KAM HO. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20:832–844, aug 1998.
- [96] B A HUBERMAN AND T HOGG. Complexity and adaptation. *Phys. D*, 2(1-3), 1986.

- [97] A. L. HUGHES AND L. PALEN. Twitter adoption and use in mass convergence and emergency events. In *Proceedings of the 6th International ISCRAM Conference*, Gothenburg, Sweden, May 2009.
- [98] H. HUSNA, S. PHITHAKKITNUKON, AND R. DANTU. Traffic shaping of spam botnets. In *Proceedings of the 5th IEEE Conference on Consumer Communications and Networking*, Las Vegas, NV, USA, January 2008.
- [99] COLLIN JACKSON, ANDREW BORTZ, DAN BONEH, AND JOHN C. MITCHELL. Protecting browser state from web privacy attacks. In *Proceedings of the 15th international conference on World Wide Web*, pages 737–744, 2006.
- [100] BERNARD J. JANSEN, MIMI ZHANG, KATE SOBEL, AND ABDUR CHOWDURY. Twitter power: Tweets as electronic word of mouth. *American Society for Information Science and Technology*, 60(11), 2009.
- [101] AKSHAY JAVA, XIAODAN SONG, TIM FININ, AND BELLE TSENG. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*, San Jose, CA, USA, 2007.
- [102] TREVOR JIM, NIKHIL SWAMY, AND MICHAEL HICKS. Defeating script injection attacks with browser-enforced embedded policies. In *Proceedings of the 16th international conference on World Wide Web*, pages 601–610, 2007.
- [103] CHRIS KANICH, CHRISTIAN KREIBICH, KIRILL LEVCHENKO, BRANDON ENRIGHT, GEOFFREY M. VOELKER, VERN PAXSON, AND STEFAN SAVAGE. Spamalytics: an empirical analysis of spam marketing conversion. *Commun. ACM*, 52:99–107, September 2009.
- [104] JUNG-HOON KIM, TAE-BOK YOON, KUN-SU KIM, AND JEE-HYONG LEE. Trackback-rank: An effective ranking algorithm for the blog search. In *Proceedings of the Second International Symposium on Intelligent Information Technology Application - Volume 03*, pages 503–507, Washington, DC, USA, 2008.
- [105] RON KOHAVI AND ROSS QUINLAN. Decision tree discovery. In *In Handbook of Data Mining and Knowledge Discovery*, pages 267–276. University Press, 1999.
- [106] BALACHANDER KRISHNAMURTHY, PHILLIPA GILL, AND MARTIN ARLITT. A few chirps about twitter. In *Proceedings of the First Workshop on Online Social Networks*, Seattle, WA, USA, 2008.
- [107] HAEWON KWAK, CHANGHYUN LEE, HOSUNG PARK, AND SUE MOON. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600, 2010.
- [108] HOMIN K. LEE, TAL MALKIN, AND ERICH NAHUM. Cryptographic strength of ssl/tls servers: current and recent practices. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 83–92, 2007.



- [109] F. LI, W. WANG, J. MA, AND H. SU. Action-based access control for web services. In *Proceedings of the 2009 Fifth International Conference on Information Assurance and Security*, volume 2, pages 637–642, 2009.
- [110] VÁCLAV MATYÁS JR. AND ZDENEK RIHA. Toward reliable user authentication through biometrics. *IEEE Security and Privacy*, 1:45–49, May 2003.
- [111] G. MCLACHLAN, K. DO, AND C. AMBROISE. *Analyzing microarray gene expression data*. Wiley, 2004.
- [112] MICROSOFT. Mitigating cross-site scripting with http-only cookies. <http://msdn.microsoft.com/en-us/library/ms533046.aspx> [Accessed: Feb. 4, 2010].
- [113] GEORGE A. MILLER. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41, 1995.
- [114] ALAN MISLOVE, MASSIMILIANO MARCON, KRISHNA P. GUMMADI, PETER DRUSCHEL, AND BOBBY BHATTACHARJEE. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, San Diego, CA, USA, 2007.
- [115] FABIAN MONROSE AND AVIEL RUBIN. Authentication via keystroke dynamics. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 48–56, 1997.
- [116] ROBERT C. NEWMAN. Cybercrime, identity theft, and fraud: practicing safe internet - network security threats and vulnerabilities. In *Proceedings of the 3rd annual conference on Information security curriculum development*, pages 68–78, 2006.
- [117] PREECHA NOIUMKAR AND THAWATCHAI CHOMSIRI. Top 10 free web-mail security test using session hijacking. In *Proceedings of the 3rd International Conference on Convergence and Hybrid Information Technology*, Oakland, CA, USA, November 2008.
- [118] ALEXANDROS NTOULAS, MARC NAJORK, MARK MANASSE, AND DENNIS FETTERLY. Detecting spam web pages through content analysis. In *Proceedings of the 15th international conference on World Wide Web*, pages 83–92, 2006.
- [119] JOON S. PARK AND RAVI SANDHU. Secure cookies on the web. *IEEE Internet Computing*, 4:36–44, July 2000.
- [120] KYOUNGSOO PARK, VIVEK S. PAI, KANG-WON LEE, AND SERAPHIN CALO. Securing web service by automatic robot detection. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 23–23, 2006.
- [121] TED PEDERSEN. Computational approaches to measuring the similarity of short contexts : A review of applications and methods. *CoRR*, abs/0806.3787, 2008.
- [122] A PORTA, G BASELLI, D LIBERATI, N MONTANO, C COGLIATI, T GNECCHI-RUSCONE, A MALLIANI, AND S CERUTTI. Measuring regularity by means of a corrected conditional entropy in sympathetic outflow. *Biological Cybernetics*, Vol. 78(No. 1), January 1998.

- [123] A. PORTA, G. BASELLI, D. LIBERATI, N. MONTANO, C. COGLIATI, T. GNECCHIRUSCONE, A. MALLIANI, AND SERGIO CERUTTI. Measuring regularity by means of a corrected conditional entropy in sympathetic outflow. *Biological Cybernetics*, 78(1):71–78, 1998.
- [124] J. R. QUINLAN. *Discovering rules from large collections of examples: A case study*. Edinburgh University Press, 1979.
- [125] G. SALTON, A. WONG, AND C. S. YANG. A vector space model for automatic indexing. *Commun. ACM*, 18:613–620, November 1975.
- [126] FABRIZIO SEBASTIANI. Machine learning in automated text categorization. *ACM Computing Surveys*, Vol. 34(No. 1), 2002.
- [127] KATE STARBIRD, LEYSIA PALEN, AMANDA HUGHES, AND SARAH VIEWEG. Chatter on the red: What hazards threat reveals about the social life of microblogged information. In *Proceedings of the ACM 2010 Conference on Computer Supported Cooperative Work*, February 2010.
- [128] ATHENA STASSOPOULOU AND MARIOS D. DIKAIKOS. Web robot detection: A probabilistic reasoning approach. *Comput. Netw.*, 53:265–278, February 2009.
- [129] BRETT STONE-GROSS, MARCO COVA, LORENZO CAVALLARO, BOB GILBERT, MARTIN SZYDLOWSKI, RICHARD KEMMERER, CHRISTOPHER KRUEGEL, AND GIOVANNI VIGNA. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and Communications Security*, Chicago, IL, USA, 2009.
- [130] LEE S. STRICKLAND. Copyright’s digital dilemma today: Fair use or unfair constraints? part 2: The dmca, the teach act and other e-copying considerations. <http://www.asis.org/Bulletin/Dec-03/strickland.html> [Accessed: May 2, 2009].
- [131] GIANLUCA STRINGHINI, CHRISTOPHER KRUEGEL, AND GIOVANNI VIGNA. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, 2010.
- [132] J. SUTTON, LEYSIA PALEN, AND IRINA SHLOVSKI. Back-channels on the front lines: Emerging use of social media in the 2007 southern california wildfires. In *Proceedings of the 2008 ISCRAM Conference*, Washington, DC, USA, May 2008.
- [133] SYMANTEC. Internet security threat report, security research and analysis. <http://www.symantec.com/business/theme.jsp?themeid=threatreport>.
- [134] PANG-NING TAN AND VIPIN KUMAR. Discovery of web robot sessions based on their navigational patterns. *Data Min. Knowl. Discov.*, 6:9–35, January 2002.
- [135] MIKE TER LOUW AND V.N. VENKATAKRISHNAN. Blueprint: Precise browser-neutral prevention of cross-site scripting attacks. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 2009.

- [136] KURT THOMAS, CHRIS GRIER, DAWN SONG, AND VERN PAXSON. Suspended accounts in retrospect: an analysis of twitter spam. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 243–258, 2011.
- [137] ALAN M. TURING. Computing machinery and intelligence. *Mind*, Vol. 59:433–460, 1950.
- [138] TWITTER. Twitter api wiki. <https://dev.twitter.com/docs> [Accessed: Aug. 28, 2011].
- [139] DE WANG, DANESH IRANI, AND CALTON PU. A social-spam detection framework. In *Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference*, pages 46–54, 2011.
- [140] GARY WASSERMANN AND ZHENDONG SU. Static detection of cross-site scripting vulnerabilities. In *Proceedings of the 30th international conference on Software engineering*, pages 171–180, 2008.
- [141] WIKIPEDIA. Inline linking (leeching, bandwidth theft). [http://en.wikipedia.org/wiki/Inline\\_linking](http://en.wikipedia.org/wiki/Inline_linking) [Accessed: May. 30, 2010].
- [142] SHAOMEI WU, JAKE M. HOFMAN, WINTER A. MASON, AND DUNCAN J. WATTS. Who says what to whom on twitter. In *Proceedings of the 20th international conference on World wide web*, pages 705–714, 2011.
- [143] MENGJUN XIE, ZHENYU WU, AND HAINING WANG. Honeyim: Fast detection and suppression of instant messaging malware in enterprise-like networks,. In *Proceedings of the 23rd Annual Computer Security Applications Conference*, Miami Beach, FL, USA, 2007.
- [144] MENGJUN XIE, HENG YIN, AND HAINING WANG. An effective defense against email spam laundering. In *Proceedings of the 13th ACM conference on Computer and Communications Security*, pages 179–190, 2006.
- [145] JEFF YAN. Bot, cyborg and automated turing test. In *Proceedings of the 14th International Workshop on Security Protocols*, Cambridge, UK, March 2006.
- [146] JEFF YAN AND AHMAD SALAH EL AHMAD. A low-cost attack on a microsoft captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 543–554, 2008.
- [147] SARITA YARDI, DANIEL ROMERO, GRANT SCHOENEBECK, AND DANAH BOYD. Detecting spam in a twitter network. *First Monday*, 15(1), January 2010.
- [148] BILL YERAZUNIS. CRM114 - the controllable regex mutilator. <http://crm114.sourceforge.net> [Accessed: Jan. 25, 2008].
- [149] JONATHAN A. ZDZIARSKI. *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification*. No Starch Press, 2005.
- [150] DEJIN ZHAO AND MARY BETH ROSSON. How and why people twitter: the role that microblogging plays in informal communication at work. In *Proceedings of the ACM 2009 International Conference on Supporting Group Work*, Sanibel Island, FL, USA, 2009.

## VITA

Zi Chu

Zi Chu received his B.E and M.E. degrees in Computer Engineering from Southeast University, Nanjing, China, in 2003 and 2006, respectively. He enrolled in the Ph.D. program of Computer Science at the College of William and Mary in 2006.

His research interests include Web applications, Internet security and privacy, statistical learning methods of abnormal behavior detection and social networking.