Towards Trustworthy Social Computing Systems

Thesis for obtaining the title of Doctor of Engineering Science of the Faculty of Natural Science and Technology I of Saarland University

From

Bimal Viswanath

Saarbrücken
April, 2016

Date of Colloquium:	28/04/2016
Dean of Faculty:	Univ.-Prof. Dr. Frank-Olaf Schreyer
Chair of the Committee:	Prof. Dr. Michael Backes
Reporters	
First Reviewer:	Dr. Krishna P. Gummadi
Second Reviewer:	Prof. Dr. Matteo Maffei
Third Reviewer:	Prof. Dr. Alan Mislove
Fourth Reviewer:	Prof. Dr. Mark Crovella
Academic Assistant:	Dr. Rijurekha Sen

ABSTRACT

The rising popularity of social computing systems has managed to attract rampant forms of service abuse that negatively affects the sustainability of these systems and degrades the quality of service experienced by their users. The main factor that enables service abuse is the *weak identity* infrastructure used by most sites, where identities are easy to create with no verification by a trusted authority. Attackers are exploiting this infrastructure to launch *Sybil attacks*, where they create multiple fake (Sybil) identities to take advantage of the combined privileges associated with the identities to abuse the system.

In this thesis, we present techniques to mitigate service abuse by designing and building defense schemes that are *robust* and *practical*. We use two broad defense strategies: (1) Leveraging the social network: We first analyze existing social network-based *Sybil detection* schemes and present their practical limitations when applied on real world social networks. Next, we present an approach called *Sybil Tolerance* that bounds the impact an attacker can gain from using multiple identities; (2) Leveraging activity history of identities: We present two approaches, one that applies anomaly detection on user social behavior to detect individual misbehaving identities, and a second approach called Stamper that focuses on detecting a group of Sybil identities. We show that both approaches in this category raise the bar for defense against adaptive attackers.

KURZDARSTELLUNG

Die steigende Popularität sozialer Medien führt zu umfangreichen Missbrauch mit negativen Folgen für die nachhaltige Funktionalität und verringerter Qualität des Services. Der Missbrauch wird massgeblich durch die Nutzung *schwacher Identifikationsverfahren*, die eine einfache Anmeldung ohne Verifikation durch eine vertrauenswürdige Behörde erlaubt, ermöglicht. Angreifer nutzen diese Umgebung aus und attackieren den Service mit sogenannten *Sybil Angriffen*, bei denen mehrere gefälschte (Sybil) Identitäten erstellt werden, um einen Vorteil durch die gemeinsamen Privilegien der Identitäten zu erhalten und den Service zu missbrauchen.

Diese Doktorarbeit zeigt Techniken zur Verhinderung von Missbrauch sozialer Medien, in dem Verteidgungsmechanismen konstruiert und implementiert werden, die sowohl robust als auch praktikabel sind. Zwei Verteidigungsstrategien werden vorgestellt: (1) Unter Ausnutzung des sozialen Netzwerks: Wir analysieren zuerst existierende soziale Netzwerk-basierende *Sybil Erkennungsmechanismen* und zeigen deren praktische Anwendungsgrenzen auf bei der Anwendung auf soziale Netzwerke aus der echten Welt. Im Anschluss zeigen wir den Ansatz der sogenannten *Sybil Toleranz*, welcher die Folgen eines Angriffs mit mehreren Identitäten einschränkt. (2) Unter Ausnutzung des Aktivitätsverlaufs von Identitäten: Wir presentieren zwei Ansätze, einen anwendbar für die Erkennung von Unregelmässigkeiten in dem sozialen Verhalten eines Benutzers zur Erkennung unanständiger Benutzer und ein weiterer Ansatz namens Stamper, dessen Fokus die Erkennung von Gruppen bestehend aus Sybil Identitäten ist. Beide gezeigten Ansätze erschweren adaptive Angriffe und verbessern existierende Verteidigungsmechanismen.

PUBLICATIONS

Parts of this thesis have appeared in the following publications.

- "Strength in Numbers: Robust Tamper Detection in Crowd Computations". Bimal Viswanath, M. Ahmad Bashir, M. Bilal Zafar, Simon Bouget, Saikat Guha, Krishna P. Gummadi, Aniket Kate, and Alan Mislove. In *Proceedings of the 3rd ACM Conference on Online Social Networks (COSN)*, Palo Alto, CA, USA, November 2015. **Best Paper Award**.

- "Towards Detecting Anomalous User Behavior in Online Social Networks". Bimal Viswanath, M. Ahmad Bashir, Mark Crovella, Saikat Guha, Krishna P. Gummadi, Balachander Krishnamurthy and Alan Mislove. In *Proceedings of the 23rd Usenix Security Symposium (USENIX Security)*, San Diego, CA, USA, August 2014.

- "Canal: Scaling Social Network-based Sybil Tolerance Schemes". Bimal Viswanath, Mainack Mondal, Krishna P. Gummadi, Alan Mislove, and Ansley Post. In *Proceedings of the 7th European Conference on Computer Systems (EuroSys)*, Bern, Switzerland, April 2012.

- "Exploring the Design Space of Social Network-based Sybil Defenses". Bimal Viswanath, Mainack Mondal, Allen Clement, Peter Druschel, Krishna P. Gummadi, Alan Mislove, and Ansley Post. In *Proceedings of the 4th International Conference on Communication Systems and Networks (COMSNETS)*, Bangalore, India, January 2012. **Invited Paper**.

- "An Analysis of Social Network-based Sybil Defenses". Bimal Viswanath, Ansley Post, Krishna P. Gummadi, and Alan Mislove. In *Proceedings of the Annual Confer-*

ence of the ACM Special Interest Group on Data Communication (SIGCOMM), New Delhi, India, August 2010.

Additional publications while at MPI-SWS.

- "Strengthening Weak Identities Through Inter-Domain Trust Transfer". Giridhari Venkatadri, Oana Goga, Changtao Zhong, Bimal Viswanath, Krishna P. Gummadi and Nishanth Sastry. In *Proceedings of the 25th International World Wide Web Conference (WWW)*, Montreal, Canada, April 2016.

- "Understanding and Specifying Social Access Control Lists". Mainack Mondal, Yabing Liu, Bimal Viswanath, Krishna P. Gummadi and Alan Mislove. In *Proceedings of the 10th Symposium On Usable Privacy and Security (SOUPS)*, Menlo Park, CA, USA, July 2014. **Distinguished Paper Award**.

- "Defending Against Large-scale Crawls in Online Social Networks". Mainack Mondal, Bimal Viswanath, Allen Clement, Peter Druschel, Krishna P. Gummadi, Alan Mislove, and Ansley Post. In *Proceedings of the 8th ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, Nice, France, December 2012.

- "Keeping Information Safe from Social Networking Apps". Bimal Viswanath, Emre Kıcıman, and Stefan Saroiu. In *Proceedings of the ACM SIGCOMM Workshop On Social Networks (WOSN)*, Helsinki, Finland, August 2012.

- "Understanding and Combating Link Farming in the Twitter Social Network". Saptarshi Ghosh, Bimal Viswanath, Farshad Kooti, Naveen Kumar Sharma, Korlam Gautam, Fabricio Benevenuto, Niloy Ganguly, and Krishna P. Gummadi. In *Proceedings of the 21st International World Wide Web Conference (WWW)*, Lyon, France, April 2012.

- "Sharing Social Content from Home: A Measurement-driven Feasibility Study". Massimiliano Marcon, Bimal Viswanath, Meeyoung Cha, and Krishna P. Gummadi. In *Proceedings of the 21st International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Vancouver, Canada, June 2011.

- "You Are Who You Know: Inferring User Profiles in Online Social Networks". Alan Mislove, Bimal Viswanath, Krishna P. Gummadi, and Peter Druschel. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining (WSDM)*, New York, NY, February 2010.

- "On the Evolution of User Interaction in Facebook". Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. In *Proceedings of the 2nd ACM SIGCOMM Workshop On Social Networks (WOSN)*, Barcelona, Spain, August 2009.

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisors, Krishna Gummadi and Alan Mislove for their help, advice and support. I am greatly indebted to them for all that I have learned from them about doing good research. I am also thankful to Alan for extending the opportunity to spend a summer at Northeastern University which was an exciting experience. I would like to thank Saikat Guha, Emre Kıcıman, and Stefan Saroiu for mentoring and providing me the opportunity to do an internship. I am also grateful to Peter Druschel and Mark Crovella for their valuable guidance and support.

I have been very fortunate to collaborate with several fantastic researchers during my graduate career. I would like to thank my co-authors, Aniket Kate, Ansley Post, Allen Clement, Balachander Krishnamurthy, Meeyoung Cha, Nishanth Sastry, Niloy Ganguly, Fabricio Benevenuto, Oana Goga, and Saptarshi Ghosh.

I am grateful to all the students who worked with me on several exciting projects. I would like to thank Mainack, Bilal, Yabing, Giridhar, Farshad, Lisette, and Max. Their immense energy and dedication fueled our various collaborative efforts. I am also fortunate to have the opportunity to mentor some very talented students, Ahmad, Advait, and Simon. I would also like to thank current and former members in the various research groups at MPI-SWS, in particular, Juhi, Riju, Przemek, Reza, Paarijaat, Ekin, Eslam, Viktor, Anjo, Arpan, Cheng, Aastha, Reinhard, Felipe, David, Vineet, Manohar, Ezgi, and Alexander for helping to make MPI-SWS a lively and exciting place to work.

I am greatly indebted to Pramod for his support and encouragement during the ups and downs of graduate school. I would also like to thank Pedro, Atul, and Animesh for the fun discussions and for making my time at work more enjoyable.

I am thankful to all the technical support staff at MPI-SWS, and in particular, Carina, and Chris for their very sincere efforts. I would also like to thank Mary-Lou for providing all the information regarding submitting the dissertation and scheduling the defense, as well as Brigitta Hansen, and Claudia Richter for their help with administrative matters.

I would like to thank my brother, Kamal for always inspiring me. I am glad to acknowledge my close friendship with Sanand, Kiran and Jithu. Their advice and support helped immensely. Finally, I am deeply thankful to my wife, Remya for all her love and support. I could not have completed this without her help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 What are social computing systems?

Social computing systems are computational systems where the workload is driven by real people through platforms that allow social interaction. A wide variety of such systems have emerged in the recent past, such as social networking services, multimedia content sharing services, blogs, wikis, social bookmarking, crowd sourcing, online gaming, social commerce and instant messaging services.

Social computing systems incorporate different social aspects of human behavior to allow people to interact, collaborate and compete with each other. Many of these systems enable users to recreate some of their offline social behavioral conventions in an online environment on the Web today. For example, users can carry out a one-on-one information exchange in the online world using social networking or instant messaging services, or they can collaborate by using group-based discussion features provided by social bookmarking or content sharing services. Popular e-commerce services such as eBay and Yelp are increasingly incorporating social features to their services where sellers can obtain endorsements from other trusted users for their transactions, similar to referrals solicited in the real world from trusted partners. Thus, social computing systems provide users a social context in the way they view and interact with content on the Web today.

In this chapter, we first start by presenting a brief history of social computing systems. Next, we provide an overview of the key architectural elements of a social computing system

to help readers understand the design of these platforms at a high level before moving to the research problem statement.

1.2 The rising popularity of social computing systems

One of the earliest social computing systems can be considered to be the Bulletin Board System (BBS) [4] launched in 1978 which allowed users to interact with each other over telephone lines. BBS software hosted on a personal computer allowed another user to dial in through the modem of the host computer to exchange information. The first instant messaging system arrived in 1988 with Internet Relay Chat (IRC) [18]. Later with the advent of the Web, more social computing sites started emerging. Geocities [31] launched in 1994 was among the first few sites that allowed users to create their own websites to put up personal information on the Web. In 1997, AOL Instant Messenger [3] was launched and instant messaging became immensely popular from then onwards. The first social networking site that allowed users to list friends, family and acquaintances was SixDegrees.com [23] which was launched in the same year.

In the coming years, several new social networking sites grew in popularity as more users became connected to the Internet. One of the early social networking sites to gain popularity was Friendster [100] (launched in 2002). On Friendster, users could share multimedia content, contact friends and friends-of-friends. Friendster became popular for dating purposes and would compete against dedicated dating websites such as Match.com. Other social networking sites were also launched during this time frame, including CyWorld [9] and LinkedIn [142] which are still very popular as of this writing. The biggest competitor for Friendster came in 2003 when MySpace [157] was launched. MySpace allowed users to customize the appearance of their *profile* which turned out to be a popular feature. Users would go on to add music, pictures and textual content to their profiles which served as a promotional space for budding musicians and artists.

As of this writing, newer social networking sites and other older social computing services continue to rise in popularity, with some newer services even eclipsing older services like Friendster and MySpace. As of September 2015, the largest social networking site is Facebook with a staggering user population of 1.55 billion monthly active users [74]. Another popular social networking platform is Google+ with 540 million monthly active users (as of October 2013) [110].[1] Twitter with 320 million monthly active users (as of September 2015) posting hundreds of millions of tweets per day is the largest micro-blogging platform where users can obtain real-time information on the Web [189]. One of the largest content sharing platforms, YouTube, has over a billion users with over 300 hours of user-generated videos uploaded to YouTube every minute [34]. Yelp, a popular social-commerce site has over 89 million monthly mobile unique visitors and over 90 million reviews written by users for different businesses around the world (as of September 2015) [207].

As social computing services continue to rise in popularity, social features are also being incorporated into other websites on the Web. *Social plugins* such as the Facebook like[2] button [19] allow users to 'like' a page on the Web and share it with their friends on Facebook. Other social networking services such as Google+, LinkedIn, Twitter also provide different types of social plugins, thus essentially enabling the new *social Web* [15, 20, 25].

1.3 Architectural elements of a social computing system

In this section, we describe the key architectural elements of a general social computing system. First, before we start examining the design of a social computing system, it is useful to briefly look at the design of the traditional Web or Web 1.0 (i.e., the first stage of the World Wide Web before social computing systems became popular) to compare and contrast with social computing systems. The Web 1.0 as shown in Figure 1.1, can be viewed as an

[1] Google's definition of an "active user" is the subject of some debate [109].

[2] When printed in this font, likes refer to Facebook "Like"s (i.e., the action of clicking on a Like button in Facebook).

information network where different pieces of Web *content* are linked or connected with each other. These links are often hyperlinks that serve as a reference from one piece of content to another that a Web reader can easily follow. The structure of such an information network embeds extremely useful information that can be leveraged for systems design. For example, the PageRank algorithm analyzes the structure of the information network to rank websites and powered the first version of the now popular Google search engine [163].

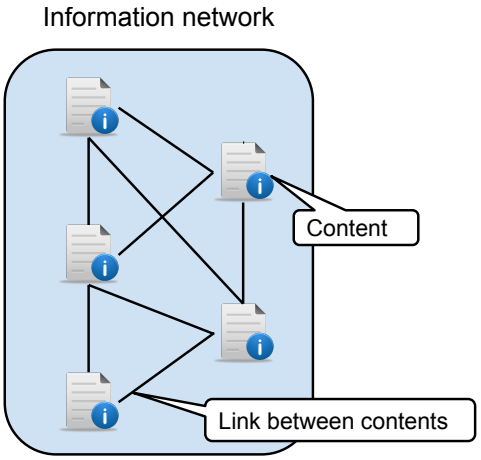


Figure 1.1: Structural design of Web 1.0.

Social computing systems on the other hand have a fundamentally different structure because of the presence of *users* or *identities* that play a key role in the system by generating and interacting with content and other users. While different social computing systems provide a variety of sophisticated services and features, at its core, most of these systems have a simple structural form underneath all the complexity. For the purpose of this thesis, we present a simplified design of a social computing system in Figure 1.2. We will be referring to this structural design in the rest of this thesis while explaining our work.

We will now present the key architectural elements of a social computing system:

Users or identities. The workload in social computing systems are driven by people who participate in the system by creating *accounts* or *identities*.[3] Users are first class entities and

[3]In the rest of this thesis, we will use the term "user" to refer to a single unique identity in the system. It is possible for a single human to create multiple identities in these systems, but we will consider such multiple

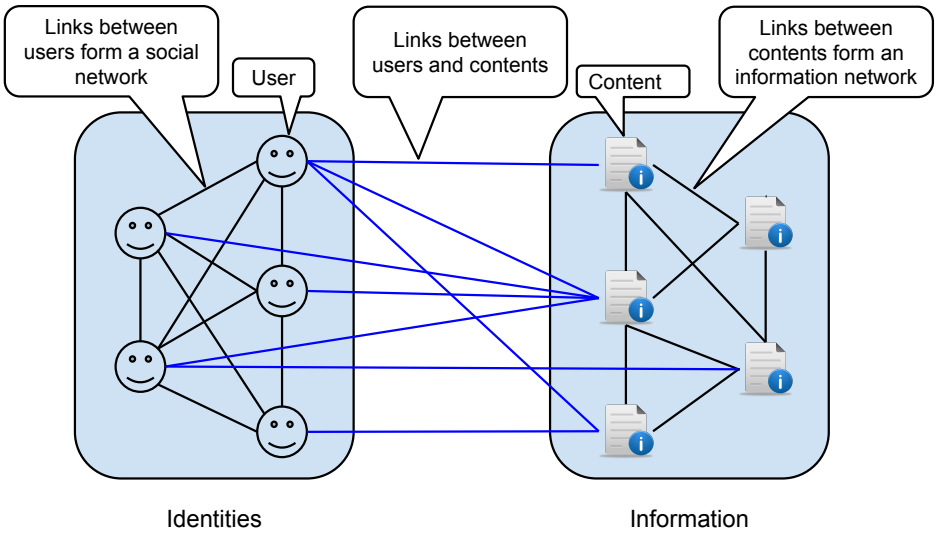


Figure 1.2: Structural design of a social computing system

add value to these systems by generating content and interacting with other content and users. Registering an account is usually free in many services and typically only require an e-mail account and full name of the user. In many services, users can create *user profiles* where they can upload basic biographical information, pictures and information about personal interests and hobbies. Note that the Web 1.0 did not have users as entities and instead was designed as a network of content as shown in Figure 1.1.

Content. Using an account in the system, users can upload different types of content including text, videos, audio clips and pictures and also tag content with relevant labels. Users are also provided *access control* features to keep their uploaded content private or share it with a select set of users (who can be friends or acquaintances) or even share it publicly to be viewed by all users of the system. It is important to note that the availability of large amounts of user-generated content usually encourages most of the interaction on these platforms. In fact, many social computing systems rely on user-generated content to attract traffic to their sites. For instance, content including reviews written for products or businesses on e-commerce sites, vacation pictures shared by a user with their friends

identities as separate users. Also, we interchangeably use the terms users and identities to refer to accounts in the system.

and family, or videos created by musicians to promote their work can all trigger further interaction on social computing services.

While user-generated content helps to drive traffic to sites, data deluge is a concern considering the sheer volume of content generated on a daily basis [188]. Hence, it is important to help users find *relevant* and *trustworthy* (i.e., avoid unwanted or spam) content. Content is usually indexed to enable easy retrieval via search interfaces and many systems also deploy sophisticated content recommendation systems [115] to enable users to find relevant content more easily.

Now that we have introduced the two key entities—content and users—we will examine the *linked* nature of these entities. Entities can be linked in different ways as described below.

Links between content. Content uploaded on social computing systems can be linked with each other, thus forming an *information network* (similar to Web 1.0) as shown in Figure 1.2. A link between two pieces of content can be formed in different ways. Two pictures can be linked because they are part of the same photo album or have similar tags. Multiple text posts by different users can be linked because they are part of the same conversation or topical thread.

Note that the similarity with Web 1.0 is limited to the presence of an information network. The next two types of links described below are unique to social computing systems.

Links between users. Users can establish links to other users. The graph formed by users is called the *social network* (see Figure 1.2). Different services enable different ways of establishing links. A link can be a *directed link* where a user establishes a link to another without the consent of the the link recipient. For example, in Twitter, a user can establish a *follow* link to another user to subscribe to that user's tweet feed. A link can be undirected when the link formation requires mutual agreement between both parties. The popular Facebook social networking service allows users to establish undirected links. For both

types of links, the recipient can be a friend, acquaintance, someone with similar interests or just someone with interesting content. The social network of a user is usually made publicly visible within a service. This means that a user can browse the friendship links of another user and visit other users' profiles by following those links, thus enabling navigation within the social network. Moreover, users are incentivized to maintain a sizable social neighborhood because many services tend to measure *influence* [56] or popularity of a user in the system as a function of the size of her social neighborhood [105].

Links between users and content. Links between users and contents model different ways in which a service allows users to interact with content. A link can be formed when a user uploads some piece of content (e.g., video, pictures, text message) to the system. When a user uploads a picture on Facebook or posts a tweet on Twitter, the service operator can associate a link between that content and the user. Links can also be formed for *content voting* activities. On e-commerce sites like Yelp, we can associate a link between a user and a business page when the user provides a rating for the business. Similarly, on Facebook, a link is formed when a user likes a page [21] or some piece of content. Many other services provide social plugins similar to the Facebook like button that allow users to express their interest for some content (e.g., a text post, picture, or a video).

We have discussed three types of links in social computing systems. Next, we will explain how system operators leverage the above design elements to enable users to find relevant and trustworthy content.

Leveraging linked entities to find relevant content. First, we discuss how we can leverage the social network (network formed by links between users) to find relevant content for users. Many social networking services (e.g., Facebook) recommends content to users based on their social neighborhood. In other words, as shown in Figure 1.3, a service like Facebook can recommend content to Alice that was uploaded or promoted by her friends (users marked by the red circles). The Facebook Newsfeed [12] application is an example of such a recommendation scheme. Thus, once a user has built up their social network by

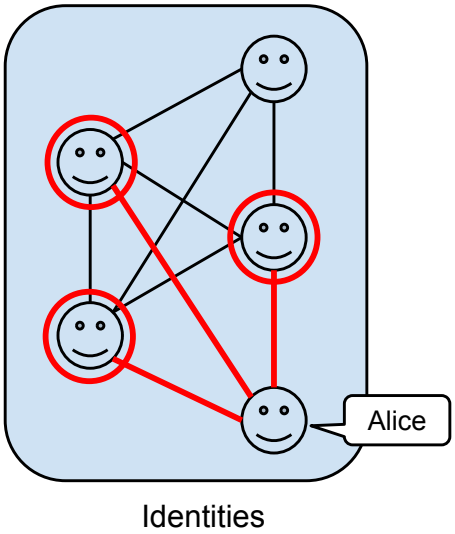


Figure 1.3: Leveraging the social network to find relevant content. User Alice can discover relevant content from her friends (users marked by the red circles)

establishing friendship links, it becomes easier for the user to find relevant and trustworthy content.

Links between users and contents is also useful for finding relevant content. Popular social networking and e-commerce sites are increasingly employing *crowd computing* to rate and rank content, users, products, and businesses. Earlier, we discussed how content rating actions map to links between users and contents. In such systems, these links can be leveraged to poll the “wisdom" or “opinions" of crowds—the users of the system—to provide a variety of recommendation services to their customers. Facebook is known to provide content recommendations to users by leveraging the number of likes received by different pieces of content (i.e., popularity of content is measured by the number of users who like the content). In Figure 1.4, if we assume that each link between an identity and content represents some type of content voting activity (e.g., Facebook like activity), user Alice can be recommended the most popular content (marked by the red circle) based on crowd opinion.

Lastly, the information network can be analyzed to find relevant groupings of content. Clustering tweets based on topical content has been shown to be useful for building recom-

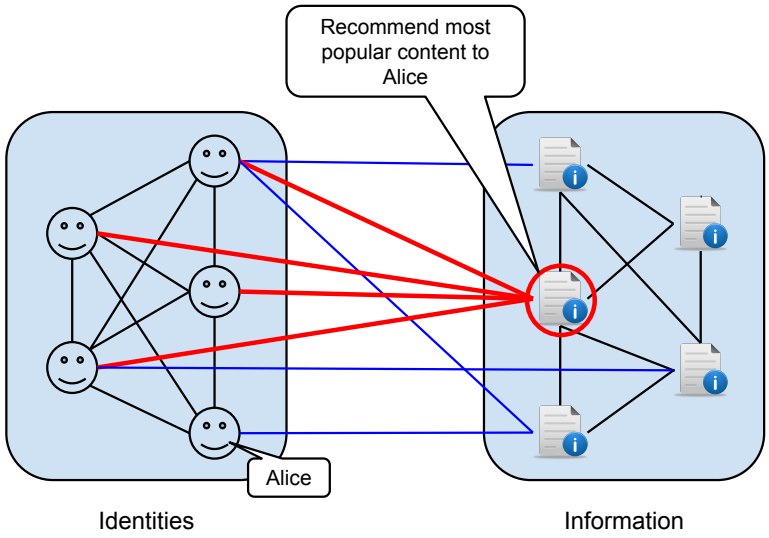


Figure 1.4: Leveraging crowd opinion to find relevant content. Links between identities and information represent content voting activity (e.g., Facebook like activity). Alice can be recommended the most popular content (marked by the red circle) based on crowd opinion.

mendation systems [53]. A large corpus of images (in services like Flickr) can be better organized for more effective information retrieval by applying network clustering algorithms that leverage tags associated with the images [43]. Note that several algorithms have already been proposed to rank content on the Web information network and sophisticated web information retrieval systems are widely used today. Similar algorithms have also proved useful for identifying relevant content in social computing systems [143].

In the next section, we introduce a key research challenge associated with social computing systems that we tackle in this thesis. Afterwards, we will present solutions that leverage the key architectural elements of these systems.

1.4 Problem: Service abuse in social computing systems

Service abuse is a serious problem. Service abuse can be broadly defined as actions taken by an attacker with the goal of manipulating or exploiting features of a social computing service, typically for financial gain. Service abuse can have a negative impact on both the

service provider and the users of the platform. Rising popularity of social computing sites has managed to attract rampant forms of *service abuse*, where attackers send spam [103, 112, 122], spread malware that can hijack user sessions [22, 191, 73], and manipulate user and content popularity/ratings [121, 162, 123]. Typically, the more popular a site, the greater the frequency and magnitude of such attacks. Attackers seek to influence the otherwise "organic" user interactions on the service. This degrades the quality of service experienced by users of these systems: spam leads to significant unwanted attention, manipulated popularity of content and users makes it difficult to find relevant and trustworthy information, and spread of malware leads to further abuse as real user accounts can be compromised and then puppeteered by the attacker. Service abuse can also have serious economic consequences for the service provider. Many popular social computing services rely on revenue generated from advertising and some have their own advertising platforms (e.g., Facebook) [11]. As of this writing, one of the largest social advertising platforms (that of Facebook) is experiencing *click-spam*, where advertisers are charged for clicks by users who have no real interest in their ads [191, 42, 76]. Click-spam reduces the trust that advertisers place on the advertising platform. Advertisers may end up leaving the platform which will negatively affect the advertising revenue stream of the social computing service.

Why are attackers targeting these systems? Financial gain is one of the main motivations for attackers who engage in service abuse [155, 125]. We present two channels frequently used by attackers for monetary gains.

First, spammers can exploit the ever increasing popularity and amount of interaction happening on these sites to obtain increased exposure for spam or malicious content. Content is known to propagate *virally* via word-of-mouth or other mechanisms on social media platforms and sometimes eventually obtaining viewership of tens of millions of users worldwide [55]. A significant fraction of users exposed to such popular content also tend to engage or interact with the content (e.g., by further promoting it or writing an opinion about the content). In fact, viral propagation is common on popular platforms like Facebook

and Twitter. A study shows that spam or unwanted content that spread through social media platforms have a high click-through rate than those that spread via email. This means that attackers now have an even better opportunity to launch large-scale spam campaigns seeking to drive traffic to online businesses that sell illicit goods or spread malicious software (malware) that might enable attackers to hijack accounts of users (to further use them for other types of service abuse).

Second, users of most social computing platforms have an incentive to improve popularity of their user profiles and content in the system and attackers can exploit this interest for financial gains. Obtaining endorsements from the crowd is highly valued in social networking platforms like Twitter and Facebook. The Facebook like button is a very popular social plugin used widely to promote content in social media. A study claims that a single Facebook like is worth $174 to brands [28]. Thus, the number of likes a Facebook page (created by a brand or person) receives can serve as a measure of popularity of the page. Similarly, businesses on Yelp are interested in improving their popularity on social media which in turn could positively affect their revenue stream. A recent marketing study shows that a one-star increase in Yelp rating (Yelp relies on a 5 star rating system) leads to a 5-9% increase in revenue for restaurants [16]. Such demand for boosting popularity and influence in social computing platforms has led to the emergence of *abuse-as-a-service*. Today, there are plenty of thriving black-market services that offer services to manipulate content and user popularity in a wide variety of popular social computing platforms for a certain fee. There are black-market sites where a user can buy 1,000 Facebook likes for as cheap as $27 [192], or 1,000 followers on Twitter for $5 [6]. Similarly, there are black-market services that provide fake reviews or ratings for businesses on Yelp and Amazon [192].

In the next section, we will discuss how attackers are exploiting these platforms.

The identity infrastructure: main source of exploitation. The main factor that enables service abuse is the identity infrastructure provided by most social computing systems, which typically allows users to operate behind *weak identities*. Weak identities do not require the

user to provide any proof or certificate (e.g., passport) backed by a trusted authority for verifying their identity at sign up time and are easy to create. Unfortunately, weak identities are easily exploited by attackers to abuse the system. A well-known attack affecting many systems today is a *Sybil attack* [72], where attackers create large number of fake (Sybil) identities and then use them to abuse the system. An alternate identity infrastructure is one based on *strong identities*, where identities are verified through certification by a trusted authority (e.g., based on passport or social security number). While a strong identity-based infrastructure could provide better security against Sybil attacks (by preventing the creation of fake accounts), most service providers do not use it because it raises the sign-on barrier for users and also raises privacy concerns for the users. Moreover, even if we moved to a strong identity-based infrastructure, it would still be hard to provide any guarantees that a verified user would never misbehave. For example, we observe cases of non-Sybil users (accounts that are not fake) being incentivized to *collude* in order to manipulate each others' popularity, and also cases where non-Sybil identities are hijacked or *compromised* by attackers to abuse the system.

At a high level, attackers are known to exploit the weak identity infrastructure in the following three ways:

Using Sybil (fake) identities. Most online sites do not require their participating identities to be bound to a real-life entity (such as a person, group, association or role therein) using *strong* (hard to forge) identities. As a result, attackers can create multiple fake (Sybil) identities and takes advantage of the combined privileges associated with these identities to attack the system. For example, in online auction systems like eBay, a fraudulent user can continue to use the system by creating a new user account whenever her existing accounts have acquired a bad reputation. Similarly, in social networking sites like Facebook or Twitter, where content is typically rated based on user feedback, an attacker can create multiple identities to cast bogus votes and manipulate content popularity. A large body

of literature has focussed on defending against Sybil attacks [194]. Sybil identities can also include *impersonated identities*, where the attacker *assumes* the identity of another real-world person [106]. The attacker might do so for various reasons, a few of which includes misleading honest users into interacting with the impersonated identity to send spam or spread malware, or for carrying out targeted social engineering attacks where the attacker tries to mislead honest users into performing certain actions or divulging confidential information.

Using compromised identities. A compromised account is a non-Sybil account (an account that is not fake) that is under the control of the attacker. Accounts can be compromised in a variety of different ways. For example, attackers are known to use phishing schemes to steal the credentials of existing non-Sybil identities [73], or use malicious browser extensions to hijack and control an identity's web sessions with the site servers [191, 126]. From an attacker's point of view, compromised accounts are valuable because they may have already acquired good reputation in the system and may also have established a network of trusted contacts in the system. Attackers can exploit these features to more effectively abuse the system, e.g., by spreading spam messages through the trusted contacts of the compromised accounts.

Using colluding identities. Another attack methodology involves non-Sybil identities who are motivated by selfish interests to collude with one another and thereby, manipulate their popularity (or popularity of their content). Some recent studies [105] focussed on how some legitimate popular Twitter users collude with one another to exchange and farm links (in a tit-for-tat "I follow you, you follow me back" scheme) and thereby, increase their perceived influence in the network. Similarly, there are online *collusion services*, where identities collude with one another to promote each others' content [191].

1.5 Thesis Research: Towards trustworthy social computing systems

The high level goal of this thesis research is to mitigate service abuse by designing and building defense schemes that are *robust* and *practical*. First, defense schemes should be robust such that it raises the bar significantly against evasive tactics and attackers are disincentivized from an economic point of view to assign more resources and effort to bypass a defense. Second, defenses should be practical so that they are deployable without requiring a clean slate-redesign of existing social computing systems and with minimal overhead for the service operators. For instance, it is unclear if existing systems will move away from a weak identity-based infrastructure in the future due to deployment barriers. Thus, we focus on techniques that do no require a change in the underlying identity infrastructure.

1.6 Related Work

In this section, we provide a brief summary of related work. More detailed description of past literature is available in the later chapters of this thesis.

Service operators require users to agree to a statement of Terms of Service (ToS) at account creation time which lists rules that users must abide by to use the service. Even though ToS agreements typically carry some implicit threats of legal action, it rarely deters attackers abusing these services. This problem has encouraged people in both industry and academia to come up with a range of technical measures to limit service abuse. We organize related work along three directions mainly inspired by how the different approaches leverage the key architectural elements of a social computing platform (described earlier in section 1.3).

Limiting creation of malicious accounts. In this direction, the focus is on the creation of the main entity in a social computing system, the identity itself. The idea here is to limit or

prevent the creation of malicious identities altogether instead of spending effort on *detection* of malicious identities after they are created. Such an approach is particularly suitable for combating Sybil attacks and typically rely on either using a strong identity infrastructure, or tying identities to resources or requiring "proof-of-work" that are hard to forge or obtain in abundance, preventing an attacker from creating many Sybil identities in the first place. We term these approaches *Sybil prevention*. For instance, Cyworld [57] requires users to present verified identities, such as passports or social security numbers, when creating new accounts. However, such approaches have not seen widespread adoption because users are typically reluctant to provide such sensitive information at account creation time. Today, many systems require proof of human time needed to create the account (i.e., solving a CAPTCHA requires a few seconds of human time). Unfortunately, with services like Freelancer [183] or Death By CAPTCHA [10], attackers can exploit the differences in the value of human time in different countries to bypass these restrictions and create a large number of Sybil identities. Other approaches include solving memory or CPU-intensive crypto-puzzles before granting access to system services [47, 35, 52]. As of this writing, Sybil prevention schemes are used by many social computing systems as the first line of defense. However, studies have shown that attackers are able to bypass these defenses and create large number of Sybil identities [183]. Note that such approaches are not going to be effective in preventing creation of compromised or colluding accounts because these accounts are typically non-Sybil accounts created by human users who have the time and resources to provide some evidence of "proof-of-work" (e.g., by solving a CAPTCHA).

Defenses that leverage the social network structure. Another direction is to leverage the social network formed by links between identities in the system. The social network is illustrated in Figure 1.2. The key assumption here is that links in a social network represent trust relationships (e.g., friendship) between users and that an identity controlled by the attacker cannot establish an arbitrarily large number of trust relationships with other honest identities. Based on this idea, researchers have explored analyzing the structure of the social

network as a mechanism for defending against Sybil attacks [210, 209, 65, 184, 166]. The most common approach is to apply graph analysis algorithms to *detect* Sybil identities in the network. We term such approaches as *social network-based Sybil detection schemes*. Once detected, the service operator can suspend the identity or remove actions taken by the identity, thereby effectively nullifying the impact of the Sybil identity's actions on the system. In our work, we explore this idea further by first understanding the limitations of existing social network-based Sybil detection schemes when applied to real world social networks (see Chapter 2). Researchers have also investigated alternate approaches to defend against Sybil attacks that leverage information about pairwise user interactions (e.g., messages exchanged between identities) in addition to the knowledge of the social network structure. Prior work includes approaches that limit unwanted communication [149] in social networking systems, Sybil attacks on content voting systems [185], and fraudulent transactions in online marketplaces [165] by analyzing both the social network structure and user interactions. In Chapter 3, we show that these existing approaches (that seem to use different techniques) can be placed under a more general category of defense schemes we call as *social network-based Sybil tolerance* [194, 195]. Instead of trying to explicitly label identities as Sybil or non-Sybil, we show that these schemes are designed to limit the impact that a Sybil attacker can have on others, regardless of the number of identities the attacker possesses. We present practical limitations of existing Sybil tolerance schemes and propose a general design template for building Sybil tolerance schemes in Chapter 3.

Defenses that leverage activity or behavioral history of identities. Defenses in this category rely on building a *behavioral profile* for identities in the service by leveraging their activity history in the system. Activity information of identities is mainly available from the links between users and content as shown in Figure 1.2. These links carry information about how an identity interacts with content in the system. Note that activity history can also include information about linking activity between users (i.e., the social network). Extensive research has explored using data mining and machine learning techniques to detect

misbehaving identities, mainly by exploiting the differences in behavior between known misbehaving and honest identities. Such techniques often rely on identifying a set of "ground truth" misbehaving and honest identities and building machine learning classifiers (e.g., decision trees or SVM classifiers) to distinguish between identities exhibiting malicious and honest behaviors [44, 122, 167, 198, 180, 179, 197]. Researchers have also proposed schemes that leverage activity history to detect malicious content (e.g., spam content in the form of text, pictures or videos) in the system [121, 102, 182]. Defense schemes in this category leverage a wide variety of activity information including linguistic characteristics of users' postings and user social behavior in the service [73, 103, 112, 141].

While many existing techniques succeed at detecting misbehaving identities, they rely on identities exhibiting particular characteristics of misbehavior. One needs to generate separate classifiers for each different type of attack method and behavior. For example, existing studies propose distinct methods for detecting Sybil [194], compromised [73], and colluding identities [105]. However, attackers can "adapt" and evade detection by changing their attack behaviors, resulting in a continual arms race between the attackers modifying the behaviors of their identities to evade detection by spam defenses and those defending the various systems striving to keep up-to-date with the latest attacker tactics. We take a step towards limiting this arms race by leveraging anomaly detection schemes and discuss a new approach along this direction in Chapter 4.

Lastly, existing techniques that try to detect individual misbehaving identities using the above approaches have a serious limitation. In many systems, it is hard to distinguish misbehaving identities (with limited activity or lacking proof-of-work) from less active, honest identities that lack proof-of-work. Attackers can take advantage of this limitation and create hard-to-detect Sybil identities to abuse services [192]. We propose a new approach called Stamper to reason about trustworthiness of identities that overcomes this limitation (see Chapter 5).

1.7 Thesis contributions

Broadly speaking, our contributions can be placed into two broad defense strategies (based on the discussion in the previous section): (1) Leveraging the social network to limit service abuse and, (2) Leveraging activity history of identities to limit service abuse.

Leveraging the social network to limit service abuse. In this space, we first try to better understand existing social network-based Sybil defense schemes and then we present an alternate approach that overcomes the limitations of existing schemes.

Analysis of social network-based Sybil detection schemes. A number of schemes have been proposed that leverage the social network to *detect* Sybils [194]. However, the research community lacked a clear understanding of how existing Sybil detection schemes compared against each other and the limitations of these schemes when applied to real-world social networks. We analyze existing Sybil detection schemes and find that, despite their considerable differences, they all work by detecting network communities (i.e., clusters of nodes more tightly knit than the rest of the graph). Our findings enabled other researchers to propose better Sybil detection algorithms and to understand the fundamental limitations of social network-based Sybil detection.

Towards social network-based Sybil tolerance. To further improve social network-based Sybil defenses, we advocate an approach called *Sybil tolerance*. We observe that what matters is not the existence of Sybil identities, but how the attacker uses Sybil identities to abuse non-Sybil identities. Based on this notion, we provide strong bounds on the impact that Sybil identities can have on non-Sybil identities in the social network. To do so, we leverage pairwise social interaction patterns between users and the structure of the social network to get around the limitations of existing Sybil detection schemes. To compare, Sybil detection schemes reason only about identities (i.e., they reason about identities being

admitted), while Sybil tolerance schemes reason about interactions (i.e., they decide whether certain interactions are allowed or denied). Thus, in a Sybil tolerance scheme, a certain pair of identities may be allowed to participate in certain interactions (e.g., sending a message) and not others, and may be allowed to interact at certain times and not others (all depending on the state of the system). Existing Sybil tolerance approaches are application specific solutions and it is unclear how we can incorporate Sybil tolerance into other types of social computing systems. Also, we show that existing approaches are not scalable when applied to large social networks and thus are not practically deployable. We designed and developed a system called Canal that can be used to make applications (with a social network) Sybil tolerant [195]. Our implementation can be deployed in practice as it scales to very large social networks with millions of users and hundreds of millions of links. In a related work, we demonstrate how we can use this framework to limit large-scale data aggregation in social networks [154].

Leveraging activity history of identities to limit service abuse. So far, the approaches we discussed require a social network between users. However, not all social computing platforms have an in-built social network or satisfy the trust assumptions made by social network-based Sybil detection schemes. This motivates the need for more generally applicable approaches to defend against abuse in social computing platforms. We discuss two approaches that mainly leverages the activity history associated with identities instead of the structure of the social network.

Detecting individual misbehaving identities. Reasoning about trustworthiness of identities becomes much harder when we consider an *adaptive attacker*, or an attacker who can mutate and change strategy. Most existing approaches to detect misbehaving identities are typically designed for a specific attacker strategy. This leads to an arms race between attackers and service operators. To limit this arms race, we propose a new approach based on *anomaly detection*, where the idea is to *learn only normal patterns of user social*

behavior and flag any behavior that deviates from normal as *anomalous*. This makes our approach capable of detecting a variety of attacks, as we do not make any assumptions about the attacker's strategy. We showed that our approach can detect Sybil, compromised, and colluding identities without any *a priori* knowledge about the attacker's strategy. Using our anomaly detector, we also investigate the Facebook social ad platform for evidence of click-fraud where we observe that a majority of clicks received for ads we ran, looked suspicious. This is one of the first studies to examine the click-fraud problem in a social ad platform.

Detecting groups of misbehaving identities. Existing defenses that largely focus on detecting individual misbehaving identities have a fundamental limitation: when a weak identity has limited or no activity history in the system, defenses lack sufficient information to determine if the identity is misbehaving or honest. In fact, many honest users also tend to have very little or no activity history and it becomes hard to distinguish between misbehaving and honest identities in such cases. We propose a new approach to mitigate abuse without explicitly detecting misbehaving identities.Our approach, called Stamper, is based on the idea that even when it is fundamentally hard to distinguish between individual Sybil and non-Sybil identities, large *groups* of Sybil and non-Sybil identities can be differentiated based on activity history. Looking at groups of identities makes sense today, because popular social computing systems are increasingly employing *crowd computing* to rate/rank content, users or businesses by polling the "wisdom" of the crowd (a group of users). For example, Yelp leverages crowd opinion to rate restaurants and it would be very helpful for the operator to know if the crowd participating in the rating computation is untrustworthy (or contains Sybil participants). Using Stamper, we have designed and deployed a publicly accessible web application called TrulyTweeting [186] that can detect tampered crowd computations in Twitter.

1.8 Organization

The rest of the thesis is organized as follows:

In Chapter 2, we present our analysis of social network-based Sybil detection schemes.

In Chapter 3, we present the idea of social network-based Sybil tolerance and the design and implementation of Canal, a system for deploying scalable Sybil tolerance schemes.

In Chapter 4, we present a technique to detect individual misbehaving identities by looking for anomalous behavioral patterns.

In Chapter 5, we present Stamper, an approach to detect groups of misbehaving identities.

In Chapter 6, we present a concluding discussion about limiting service abuse in social computing systems.

CHAPTER 2

Analysis of social network-based Sybil detection schemes

Sybil attacks [72] pose a fundamental problem in social computing systems. Malicious attackers can create multiple fake (Sybil) identities and influence the working of systems that rely upon a weak identify infrastructure. Traditional defenses against Sybil attacks rely on trusted identities provided by a certification authority. But requiring users to present trusted identities runs counter to the open membership that underlies the success of social computing systems systems in the first place.

Recently, there has been excitement in the research community about applying social networks to mitigate Sybil attacks. A number of social network-based *Sybil detection schemes* have been proposed that attempt to detect identities that are likely to be Sybil by using properties of the social network's structure [210, 209, 65, 185, 194]. Unlike traditional solutions, these schemes require no central trusted identities, and instead rely on the trust that is embodied in existing social relationships (i.e., social network links) between users.

All social network-based Sybil detection schemes make the assumption that, although an attacker can create arbitrary Sybil identities in social networks, he or she cannot establish an arbitrarily large number of social connections to non-Sybil identities [194]. Under this assumption, Sybil identities tend to be poorly connected to the rest of the network, compared to the non-Sybil identities. Sybil detection schemes leverage this observation to detect Sybils.

They use various graph analysis techniques to search for topological features resulting from the limited capacity of Sybils to establish social links.

Our focus in this chapter is on the graph analysis algorithms behind the schemes. Most papers on Sybil detection schemes describe new algorithms, but none provide a common insight that explains how all of these schemes are able to detect Sybils. Each algorithm has been shown to work well under its own assumptions about the structure of the social network and the links connecting non-Sybil and Sybil identities. However, it is unclear how these algorithms would compare against each other, on more general topologies, or under different attack strategies. As a result, it is not known if there exist other (potentially better) ways to detect Sybils or if there are fundamental limits to using only the structure of the social network to defend against Sybils.

We take a first, but important, step towards answering these questions. We decompose existing Sybil detection schemes and demonstrate that at their core, the various algorithms work by implicitly *ranking* nodes (vertices in the social graph that represent identities) based on how well the nodes are connected to an *a priori* known trusted node. Nodes that have better connectivity to the trusted node are ranked higher and are deemed to be more trustworthy. We show that, despite their considerable differences, all Sybil detection schemes rank nodes similarly—nodes within *local communities* (i.e., clusters of nodes more tightly knit than the rest of the network) around the trusted node are ranked higher than nodes in the rest of the network. Thus, Sybil detection schemes work by effectively detecting local communities.

The above insight has important implications for both existing and future designs of social network-based Sybil defense schemes. First, it motivates us to investigate whether a class of algorithms, known as *community detection* algorithms [99], that attempt to find such clusters of nodes directly, could be used for Sybil detection. We find that it is possible to use off-the-shelf community detection algorithms to find Sybils. Unlike Sybil detection, community detection is a well-studied and mature field, implying that our findings open

the door for researchers to exploit a variety of techniques from a rich body of community detection literature [99].

Second, our insight also hints at the limitations of relying on communities for finding Sybils. For Sybil detection schemes to work well, all non-Sybil nodes need to form a single community that is distinguishable from the group of Sybil nodes.[1] In reality, however, users in many social networks form multiple communities that are interconnected rather sparsely. We show that, in these networks, it is hard for a trusted node to distinguish Sybils from non-Sybils outside its local community. Further, we demonstrate how Sybils can launch extremely effective attacks by establishing just a small number of links to carefully targeted nodes within such networks.

2.1 Understanding Sybil detection

As noted before, a variety of Sybil detection schemes have been proposed, but each has been evaluated using different social networks and attack strategies by the Sybils. Therefore, it is not well understood how these different schemes compare against each other, or how a potential user of these schemes, such as a real-world social networking site, would select one scheme over another.

2.1.1 The core of Sybil detection schemes

Given the problem of comparing competing Sybil defense schemes, one approach would be to view the schemes as complete coherent proposals (i.e., treat them as black boxes, and compare them in real-world settings). Such an approach is straight-forward and would provide useful performance comparisons between a *fixed* configuration of schemes over a *given* set of social networks and attack strategies by the Sybils. However, it would not yield

[1] Many Sybil detection schemes impose this requirement implicitly by assuming that the non-Sybil region of the network is *fast mixing* [151], meaning a random walk of length $O(\log N)$ reaches a stationary distribution of nodes.

conclusive information on how a particular scheme would perform if either the given social network or the behavior of the attacker should change. It also does not allow us to derive any fundamental insights into how these schemes work, which might enable us to build upon and improve them.

An alternative approach is to find a core insight common to all the schemes that would explain their performance in *any* setting. Gaining such a fundamental insight, while difficult, not only provides guidance on improving future designs, but also sheds light on the limits of social network-based Sybil detection. However, we cannot gain such an insight by treating each of these schemes as a black box, with each carrying its own set of algorithms, optimizations, and assumptions. Instead, we need to reduce the schemes to their core task before analyzing them.

At a high level, all existing schemes attempt to isolate Sybils embedded within a social network topology. Every scheme declares nodes in the network as either Sybils or non-Sybils from the perspective of a *trusted node*, effectively partitioning the nodes in the social network into two distinct regions (non-Sybils and Sybils). Hence, each Sybil detection scheme can actually be viewed as a *graph partitioning algorithm*, where the graph is the social network. However, the quality and performance of the algorithm depends on the inputs, namely, the network topology and the trusted node.

Most Sybil detection schemes include a number of useful and practical optimizations that enhance their performance in specific application scenarios. For example, SybilGuard [210] and SybilLimit [209] have a number of design features that facilitate their use in decentralized systems. Similarly, SumUp [185] has optimizations specific to online content voting systems. However, because our goal is to uncover the core graph partitioning algorithm, we study these schemes independent of the assumptions about their application environments as well as the optimizations that are specific to those environments. Later in the chapter, we show that this approach not only offers hints for the designers of future

Sybil detection schemes, but also helps us understand the characteristics of real-world social networks that make them vulnerable to Sybil attacks.

2.1.2 Converting partitions to rankings

Even when viewing the schemes as graph partitioning algorithms, comparing the different Sybil detection schemes is not entirely straightforward. The output of each scheme depends on the setting of numerous parameters. At a high level, these parameters can be seen as making the partitioning between Sybils and non-Sybils either more restrictive or permissive, thereby trading false positives for false negatives. While the designers of the schemes offer rough guidelines for choosing the parameter values (e.g., set a parameter to $O(\log N)$ where N is the number of network nodes), there can be considerable variation in the output from different parameter settings that follow the guidelines. Given the difficulty in selecting the right parameter settings, we would like to compare the schemes independent of the choice of their respective parameters.

We studied the impact of changing parameters on the output of the Sybil and non-Sybil partitions. We observed that as the Sybil partition grows or shrinks in response to parameter changes, an ordering can be imposed on the nodes added or removed.[2] That is, when the Sybil partition grows larger, new nodes are added to the partition without removing nodes previously classified as Sybils. Similarly, when the Sybil partition grows smaller, some nodes are removed from the partition without adding any nodes previously classified as non-Sybils. Figure 2.1 illustrates how different partitionings obtained by changing parameters can be converted into an ordering or ranking of nodes.

Our observation suggests that one can view the Sybil detection schemes as implicitly ordering or *ranking* nodes in the network, while the parameter settings determine where the boundary between the partitions, called the *cutoff point*, lies. Changing the parameters slides

[2]While we do not formally prove that all parameters of any Sybil detection scheme must induce an ordering, it is the case for all schemes, environments, and parameters we analyzed.

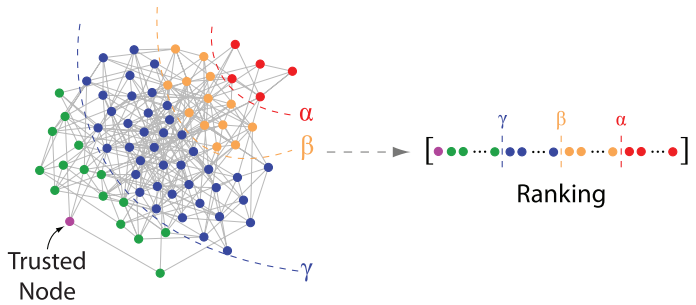


Figure 2.1: Diagram of converting partitionings into a ranking of nodes. Different parameter settings (α, β, γ) cause increasingly large partitions to be marked as Sybils, thereby inducing a ranking.

the cutoff point along the ranking, but the resulting partitions uphold the observed ranking of nodes. Thus, we can compare the different schemes independently of their parameters by simply comparing their relative rankings of the nodes.

2.1.3 Reduction of existing schemes

We reduce each Sybil detection scheme into its component processes using the model presented in Figure 2.2. At its core, each scheme contains an algorithm, which, given a trusted node and a network, produces a ranking of the nodes in the network relative to the trusted node. Then, depending on the setting of various parameter values, the scheme creates a cutoff, which is applied to the ranking and produces a Sybil/non-Sybil partitioning.

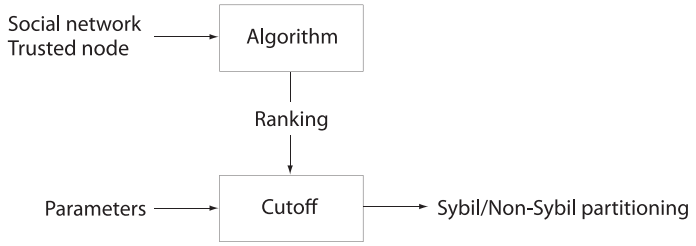


Figure 2.2: Diagram showing the processes involved in a Sybil detection scheme. In brief, the scheme itself can be split into an algorithm, which when given a social network and a trusted node, produces a ranking. The parameters to the scheme are used to create a cutoff, which defines a Sybil/non-Sybil partitioning from the ranking.

The schemes that we examine in this chapter are SybilGuard [210], SybilLimit [209], SybilInfer [65], and SumUp[3] [185]. For each of these Sybil defense schemes, Table 2.1 identifies the partitioning algorithm, how this partitioning induces a ranking of nodes, and how the algorithm parameters determine a cutoff. We also describe the assumptions the schemes make about their input environment (i.e., the structure of non-Sybil and Sybil topologies), and briefly describe the networks that these schemes were evaluated upon. A more detailed description of how these schemes map into our model is included in Appendix A.

Although we only show how our model applies to four well-known schemes, we believe that it could be applied to other schemes as well. For example, another work proposes a Sybil-resilient distributed hash table routing protocol [138, 137], by using social connections between users to build routing tables. The protocol relies on random walks much in the same manner as SybilGuard and SybilLimit, so we believe our analysis would apply to it as well. Similarly, Quercia et al. [166] proposed a Sybil detection scheme that relies on a graph-theoretic metric called betweenness centrality to calculate the likelihood of a node being a Sybil. To apply our analysis, the centrality measure can be used directly to induce a ranking of the nodes. Also, since our work in this chapter was published [196], other researchers have proposed newer social network-based Sybil detection algorithms [50, 152, 49, 184, 201, 174, 205, 48], and some of those approaches leverage the findings of our work [49, 201, 174, 48].

2.1.4 Rest of the chapter

In this section, we have shown that existing Sybil detection schemes all work by inducing an implicit ranking of the nodes. We now take a closer look at these rankings, using them to

[3]Note that strictly speaking, SumUp was not originally designed as a Sybil detection scheme, instead it was proposed to limit manipulation of content ratings by Sybils and is also discussed as a *Sybil tolerance scheme* in Chapter 3. We include SumUp in our analysis because it has characteristics similar to other Sybil detection schemes and induces a ranking of nodes based on the likelihood of being Sybil.

	Assumptions	Algorithm	Ranking	Cutoff	Evaluation
SybilGuard [210]	Non-Sybil region is fast mixing [151]	Random walk performed by each node	Varying random walk length	Whether or not walk intersection occurs	Kleinberg net-work [127]
SybilLimit [209]	Non-Sybil region is fast mixing	Multiple random walks performed by each node	Varying number of random walks and walk length	Whether or not tails of random walks intersect	Friendster, LiveJournal, DBLP, Kleinberg
SybilInfer [65]	Non-Sybil region is fast mixing, modified walks are fast mixing	Bayesian inference on the results of the random walks	Probability of node being non-Sybil from Bayesian inference	Threshold on the probability that a given node is non-Sybil	Power-law net-work [158], LiveJournal
SumUp [185]	Non-Sybil region is fast mixing, no small cut between collector and non-Sybil region	Creation of voting envelope with appropriate link capacities around collector	Varying the size of the voting envelope	Whether or not nodes are within the voting envelope	YouTube, Flickr, Digg

Table 2.1: Overview of the properties and evaluation of social network-based Sybil detection schemes.

compare the schemes across a wide range of conditions. Our goal in the remaining sections is to better understand the ranking algorithms underlying existing Sybil detection schemes, and through this understanding, to provide a basis for answering the following questions:

- Are the different Sybil detection schemes performing the core task of ranking nodes in the same way, or is each ranking unique? (Section 2.2)
- Are there other (potentially better) ways to obtain these node rankings? (Section 2.3)
- What structural properties of the social network determine how well the schemes work? (Section 2.4.1)

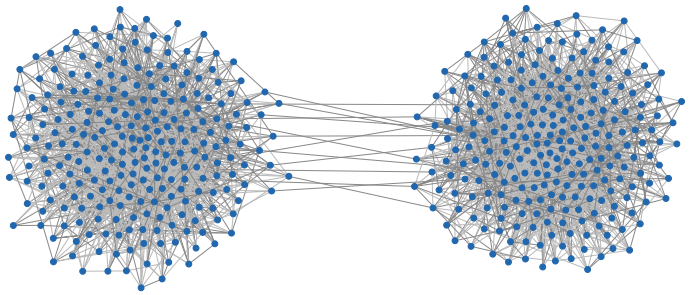


Figure 2.3: The synthetic network used in Section 2.2.1 for exploring the rankings. Each of the two communities contains 256 nodes.

- Are the schemes robust against the different possible Sybil attack strategies? (Section 2.4.2)

2.2 Rankings and Sybil detection

In this section, we develop a better understanding of the process by which Sybil detection schemes compute node rankings by comparing the rankings of the different schemes.

2.2.1 Rankings in synthetic networks

We start by examining the node rankings generated by the schemes when run over a synthetic network topology, taken from [40] and shown in Figure 2.3. In brief, this network is constructed using the Bárabási-Albert preferential attachment model [41], and then rewired[4] to have two densely connected communities of 256 nodes each, connected by a small number of edges.

[4]In brief, the rewiring works as follows: Nodes are first randomly assigned to two communities. Then, rewiring works by selecting two links $A \leftrightarrow B$ and $C \leftrightarrow D$ where A and C are in the same community and B and D are in the same community. These two links are replaced with the links $A \leftrightarrow C$ and $B \leftrightarrow D$, thereby increasing the intra-community links without changing the degree distribution or link count.

2.2.1.1 Comparing node rankings

We randomly selected a node in one of the communities as the trusted node and calculated the node rankings on this synthetic network for the four Sybil detection schemes previously discussed. We then examined how closely the various rankings matched. To compare the rankings, we use mutual information [177], which measures the similarity of two partitionings of a set. In brief, mutual information ranges between 0 and 1, where 0 represents no correlation between the partitionings, and 1 represents a perfect match.

The results of this experiment are shown in the top graph of Figure 2.4. For clarity, we only show the mutual information between partitionings of SybilGuard and each of the other three schemes (the other pairs are similar). The x-axis denotes the size of the partition containing non-Sybils. For example, the x-axis value of 10 divides the ranking into two parts, one with the first 10 nodes in the ranking (marked as non-Sybils) and the other with the rest of the nodes (marked as Sybils). Thus, Figure 2.4 shows the mutual information between pairs of rankings at all possible cutoff points.

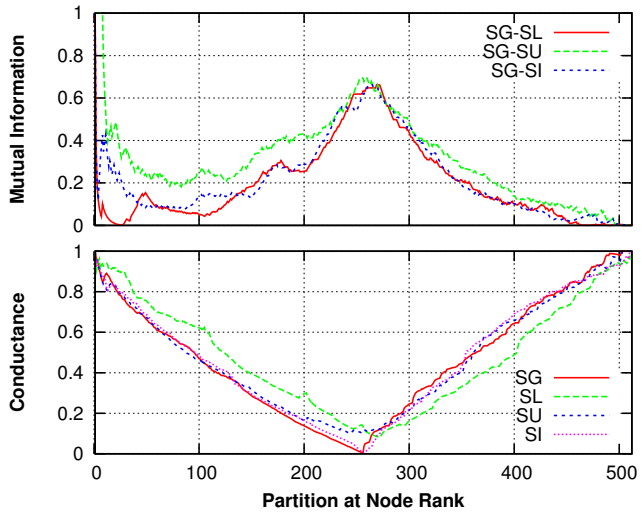


Figure 2.4: Mutual information between pairs of rankings and conductance of each ranking plotted for various partitions for the synthetic network, using schemes SybilGuard (SG), SybilLimit (SL), SumUp (SU), and SybilInfer (SI). A strong correlation is observed at 256 nodes, indicating a high degree of overlap between the partitionings, and a strong community structure in the non-Sybils, at this point.

Figure 2.4 shows that the mutual information metric is maximized at a partitioning of size 256. Interestingly, it falls off sharply before and after this cutoff value. To understand this plot better, we investigated the strong correlation between the different node rankings at the partitioning size of 256 and found that the 256 members that each scheme assigned to the non-Sybil partition strongly corresponded to the half of the network in Figure 2.4 that contained the trusted node. This indicates that all schemes are biased towards ranking nodes in the local community around the trusted node higher than nodes outside of the community. However, there is little correlation between the ordering of nodes within the community, or the nodes outside of it, as the mutual information is low between pairs of rankings before and after this point.

2.2.1.2 The common factor behind the rankings

One hypothesis that could explain our above observations is that the nodes are being ranked such that nodes well connected to the trusted node are more likely to be higher in the rankings. Since there are several nodes within the local community of the trusted node that are equally well connected, the ranking amongst these nodes is not strictly enforced, i.e., the different schemes rank these nodes differently. Similarly, several nodes outside the local community are equally poorly connected and so their relative ranking is not consistent across the different Sybil schemes. However, there is a sharp distinction between the connectivity of nodes inside and outside the local community, and so the former are ranked before the latter.

To confirm this hypothesis, we used a well known metric called *conductance* [136] for determining how closely a subset of nodes within a network are connected among themselves relative to the rest of the network. Conductance is a widely used metric for evaluating the quality of communities within large networks. In brief, the conductance of a set of nodes ranges between 0 and 1, with lower numbers indicating stronger communities.

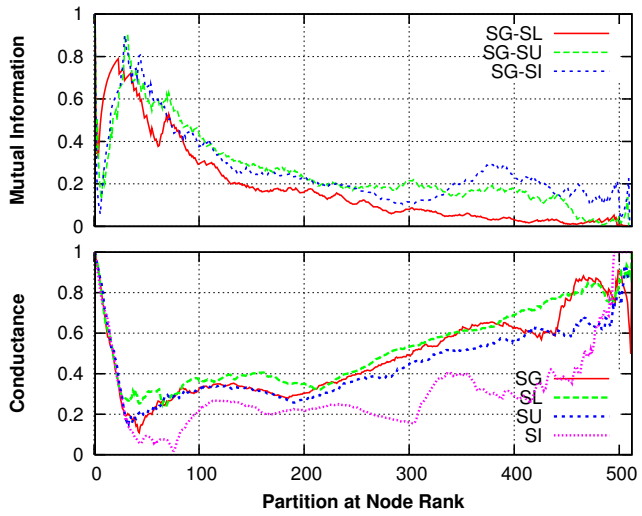


Figure 2.5: Mutual information between pairs of rankings and conductance of each ranking plotted for various partitions of the four schemes when run on the Facebook network.

We plot the conductance of the non-Sybil subset in the bottom of Figure 2.4 and notice that there is a sharp inflection point in the conductance at 256 nodes for all schemes. This corresponds to the boundary between the two communities in our synthetic network topology. Adding nodes from another community sharply increases the conductance, so all schemes assign higher rankings to nodes from within the community around the trusted node than to nodes from outside the community. This helps explain why the partitions obtained from the rankings match very well when the cutoff is set at the inflection point.

2.2.2 Rankings in real-world networks

In this section, we verify that the results we found for our synthetic network also hold in real-world networks. First, we wish to check that nodes are ranked in a biased manner, such that nodes from the trusted node's local community rank higher than any other nodes. Second, we wish to test if the point at which all Sybil detection schemes agree corresponds to a trough in the conductance value, indicating the boundary of the community around the trusted node.

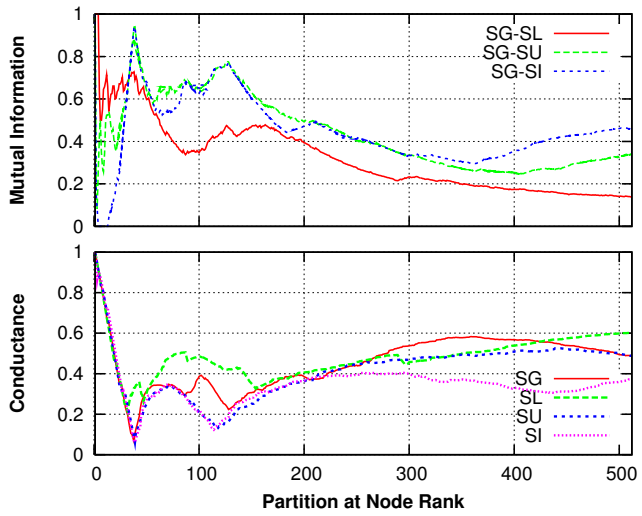


Figure 2.6: Mutual information between pairs of rankings and conductance of each ranking plotted for various partitions of the four schemes when run on the Astrophysics network.

To show this, we repeat the experiment above for two real world networks: *Facebook*, consisting of the social network between Rice University graduate students taken from Facebook [150], and *Astrophysics*, consisting of the co-authorship network between astrophysicists [160]. Details on these datasets are provided in Table 2.2.

Network	**Nodes**	**Links**	**Avg. degree**
YouTube [149]	446,181	1,728,938	7.7
Astrophysicists [160]	14,845	119,652	16
Advogato [36]	5,264	43,027	16
Facebook [150]	514	3,313	13

Table 2.2: Statistics of datasets used in our evaluation.

As we can see in Figures 2.5 and 2.6, the mutual information reveals a local cutoff where all rankings have strong correlation, and this cutoff is also characterized by a low conductance value. Taken together, our experiments show that all Sybil detection schemes are identifying a local community that surrounds the trusted node, but that the ranking of nodes they use to reach the local community (and that they use after this point) is not strongly correlated.

2.2.3 Summary of observations

We now summarize the findings from our comparison of the way in which various algorithms rank nodes:

- The ranking of nodes is biased towards those which decrease conductance. Thus, nodes that are tightly connected around a trusted node (i.e., those that form subsets with lower conductance) are more likely to be ranked higher.
- When there are multiple nodes that are similarly well connected to the trusted node (i.e., they form subsets with similar conductance) they are often ordered differently in different algorithms.
- When the trusted node is located in a densely connected community of nodes, with a clear boundary between this community and the rest of the network, the nodes in the local community around the trusted node are ranked before others.

2.3 Applying Community Detection

In the previous section, we observed that all Sybil detection schemes work by identifying nodes in the local community around a given trusted node and ranking them as more trustworthy than those outside. In this section, we examine whether algorithms that are explicitly designed to detect communities, called *community detection* algorithms [38, 58, 40, 144], can be used for Sybil detection in the same manner as existing schemes. Our goal is to investigate the potential for leveraging existing literature in community detection to defend against Sybils. To this end, we first select an off-the-shelf community detection algorithm and generate a node ranking from the algorithm. We then compare its node ranking with those of existing Sybil detection schemes, to determine if it is able to defend against Sybils with similar accuracy.

2.3.1 Community detection

Community detection in networks is a well studied and mature field. There are numerous approaches that use different mechanisms in order to detect communities and different metrics to evaluate the quality of communities. Below, we give a brief overview of how community detection schemes work.

In this chapter, we focus on *local* community detection schemes [40], which do not require a global view of the network.[5] Most of the local approaches work by starting with one (or more [38]) seed nodes and greedily adding neighboring nodes until a sufficiently strong community is found. For example, Mislove's algorithm[150] iteratively adds nodes that improve the the normalized conductance (a metric closely related to conductance) at each step, and stops when the conductance metric reaches an inflection point. For a detailed survey of local community detection algorithms, we refer the reader to the survey paper by Fortunato [99], which discusses numerous algorithms for community detection.

As there is a large body of work on community detection, we could theoretically utilize any of these algorithms as the ranking algorithm. For the evaluation presented in this section, we selected Mislove's algorithm [150], but with the conductance metric from Section 2.2.1.2. We chose this algorithm as it is conceptually easy to understand, since it greedily minimizes conductance. However, our decision is not fundamental, and there may be other algorithms that perform better (especially since different community detection algorithms have been shown to perform better on different networks [135]). Rather, our goal here is simply to investigate how well off-the-shelf community detection algorithms are able to find Sybils.

In order to use community detection to find Sybils, we need to generate a node ranking in the same manner as the other schemes. To do so, we run Mislove's community detection algorithm and record the node that it iteratively adds at each step to minimize conductance.

[5] Our decision to focus on local community detection algorithms, as opposed to global ones, is due to the fact that they work in a similar manner as existing Sybil detection schemes by not assuming a global view. However, it has been shown that different global community detection algorithms have many of the same properties as local ones [135], indicating that our results would likely hold for global algorithms as well.

Note that we modify the algorithm to not stop once a local trough is found; instead we allow it to continue running until all of the nodes have been added. This results in a node ranking that we can use to compare against the other schemes.

2.3.2 Evaluating Sybil detection

We now evaluate the community detection algorithm against our existing Sybil detection schemes. When comparing against each of the Sybil detection schemes, we used experimental settings similar to those described in the paper in which the scheme was proposed. This required us to split our evaluation results in two separate sections; one for SybilGuard, SybilLimit, and SybilInfer and another for SumUp. The split is necessary because SumUp was originally evaluated for its ability to limit the number of votes Sybil identities can place, and not for its ability to accurately detect Sybil nodes. Thus, the experimental settings for evaluating SumUp are quite different from those of the other schemes, necessitating a separate evaluation.

A summary of the data sets that we use in the evaluation is shown in Table 2.2. In addition to the datasets from the previous section, we examine *YouTube*, consisting of the social network of users in YouTube [149], and *Advogato*, consisting of the trust network between free software developers [36].

2.3.2.1 Measuring Sybil detection accuracy

In order to measure the accuracy of the various schemes at identifying Sybils, we need a way to compute how often a scheme ranks Sybil nodes towards the bottom of the ranking. To do so, we use the metric *Area under the Receiver Operating Characteristic (ROC) curve* or A'. In brief, this metric represents the probability that a Sybil detection scheme ranks a randomly selected Sybil node lower than a randomly selected non-Sybil node [97]. Therefore, the A' metric takes on values between 0 and 1: A value of 0.5 represents a random ranking, with higher values indicating a better ranking and 1 representing a perfect non-Sybil/Sybil

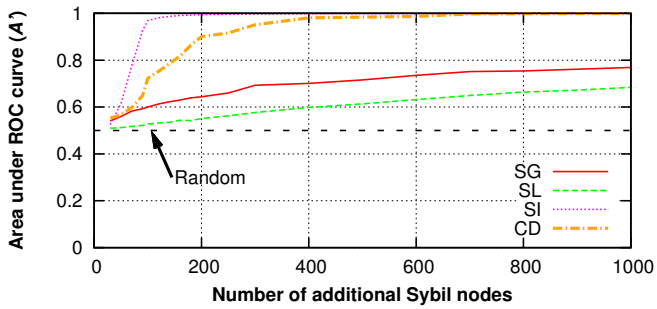


Figure 2.7: Accuracy for Sybil detection schemes, as well as community detection (CD), on the synthetic topology as we vary the number of additional Sybil identities introduced by the adversary.

ranking. Values below 0.5 indicate an inverse ranking, or one where Sybils tend to be ranked higher than non-Sybils. A very useful property of this metric is that it is defined independent of the number of Sybil and non-Sybil nodes, as well as the cutoff value, so it is comparable across different experimental setups and schemes.

2.3.2.2 SybilGuard, SybilLimit, and SybilInfer

For comparing SybilGuard, SybilLimit, and SybilInfer to the community detection algorithm, we use the same experimental methodology as the most recent proposal, SybilInfer. Specifically, we use a 1,000 node scale-free topology [41] for the non-Sybil part of the network. Among this set of non-Sybil nodes, we assume that a small fraction (10%) of the nodes belong to an adversary and become Sybil nodes. These 100 malicious nodes are chosen uniformly at random. These nodes then introduce additional Sybil identities into the network, which form a scale free topology among themselves using the same parameters as non-Sybil region. We vary the number of introduced nodes from 30 to 1,000, and average the results over 100 experimental runs.

We present the results of this experiment in Figure 2.7. We make two important observations: First, SybilInfer and community detection perform well, with improving accuracy as more Sybils are added. The reason for this increase is that the Sybil region becomes larger and, therefore, easier to distinguish from the non-Sybil region. Second, both

SybilGuard and SybilLimit perform less well than the other two schemes. This effect is because the number of Sybil nodes added is lower than the bound enforced by these two schemes, as was observed in the evaluation on SybilInfer [65]. In more detail, the Sybil region is connected to the non-Sybil region by 789 attack edges on the average; SybilGuard and SybilLimit ensure that no more that $O(\log N)$ nodes will be accepted per attack edge, where N is the number of nodes in the network. Since we only add a maximum of 1,000 Sybil nodes, neither of these schemes marks many nodes as Sybils.

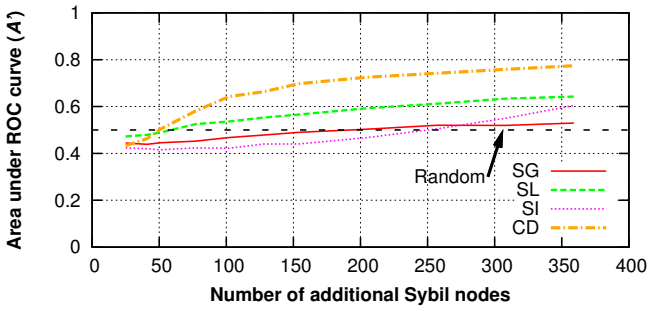


Figure 2.8: Accuracy in the Facebook network as we vary the number of additional Sybil identities introduced by the adversary.

We now evaluate these schemes on a real-world social network. Specifically, we repeat this experiment on the Facebook graduate student network from before. This network has similar density as the synthetic network, but is only half the size. The results of this experiment are presented in Figure 2.8. As we can see, the community detection algorithm performs favorably compared to the explicit Sybil detection schemes, and all become more accurate as more Sybils are added. A careful reader may note that the absolute accuracy of all schemes (community detection included) is significantly lower than that observed above in Figure 2.7. The underlying reason for this lower performance is a structural characteristic of the Facebook network that makes it inherently harder to distinguish Sybils from non-Sybils. We explore this limitation in greater detail in Section 2.4.

2.3.2.3 SumUp

Recall that SumUp provides a Sybil-resilient voting service. To do so, SumUp defines a *voting envelope* wherein the links are assigned a capacity so that all votes from within the envelope can be collected. Outside this envelope, votes are only collected if the voter can find a path with capacity to the vote collector (i.e., the trusted node). In order to apply community detection, we replace the process that determines the voting envelope with a community detection algorithm, pick the community with the lowest conductance value to be the envelope, and unconditionally accept all votes from nodes within this envelope. For nodes outside the envelope, we assign all other links to have capacity one, and we collect their votes if they can find a path with weight to any node within the envelope. This difference is necessary since we don't assign weights to links within the envelope, as SumUp does.

We evaluate and compare the community detection scheme against SumUp on three different datasets: Advogato, Astrophysics, and YouTube. We follow the same methodology used in the original SumUp evaluation [185]: for each network, we inject 100 attack edges by inserting 10 Sybil nodes with links to 10 other uniformly randomly chosen non-Sybil nodes. In order to cast bogus votes, each Sybil node is further attached to a large number of Sybil identities by a single link each. As in the original evaluation, we randomly select a vote collector and randomly choose a subset of non-Sybils as voters. We plot the average statistics over five experimental runs for both SumUp and the community detection algorithm.

To evaluate the accuracy of these schemes, we must define a new metric. This is because SumUp does not classify all nodes as Sybil or non-Sybil (needed for A'), but rather, only those nodes which issue votes. Since subsets of both the non-Sybil and Sybil nodes are issuing votes, ideally, the scheme would only count the non-Sybil votes. Thus, our metric should penalize the under counting of non-Sybil votes, as well as the counting of any Sybil votes. The metric we define, *vote accuracy*, is expressed as the number of non-Sybil votes counted divided by the sum of the number of non-Sybil votes issued and the number of

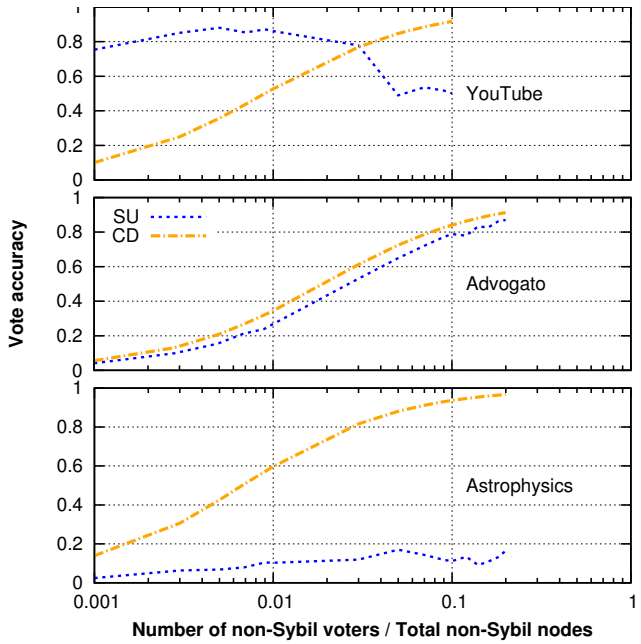


Figure 2.9: Vote accuracy of SumUp and community detection on three networks.

Sybil votes counted. Vote accuracy ranges between 0 and 1, where higher values represent better performance.

Figure 2.9 presents the results of this experiment, as we vary the number of non-Sybil voters (Sybils try to vote as often as they can). The most salient result is that the accuracy for SumUp varies widely across the three networks; this is a direct result of using the envelope technique. In certain networks, one or more of the Sybil nodes is accepted into the envelope, and a large number of malicious votes are cast. The results for the community detection algorithm are significantly more stable, producing useful results once the number of non-Sybil voters rises above 1%.

2.3.3 Implications

We began this section by observing that, since all Sybil detection schemes appeared to be identifying local communities, explicit community detection algorithms may be able to defend against Sybils as well. It is interesting to note—even without changing the experimental setup under which existing schemes were evaluated—our simple community

detection algorithm gives comparable results to existing schemes. Our results have both positive and negative implications for future designers of Sybil defense schemes.

On the positive side, our results demonstrate that there is an opportunity to leverage the large body of existing work on community detection algorithms for Sybil detection [99]. Prior work on community detection provides a readily available source of sophisticated graph analysis algorithms around which researchers could improve existing schemes and design new approaches. On the negative side, relying on community detection for performing Sybil detection fundamentally limits the ability of these schemes to find Sybils in many real-world graphs. We explore these limitations in the next section.

2.4 Limitations of social network-based Sybil detection

In the previous sections, we showed that Sybil detection schemes work by effectively identifying nodes within tightly-knit communities around a given trusted node as more trustworthy than those farther away. In this section, we investigate the limitations of relying on community structure of the social network to find Sybils. More specifically, we explore how the structure of the social network impacts the performance of Sybil detection schemes and how attackers with knowledge of the structure of the social network can leverage it to launch more efficient Sybil attacks.

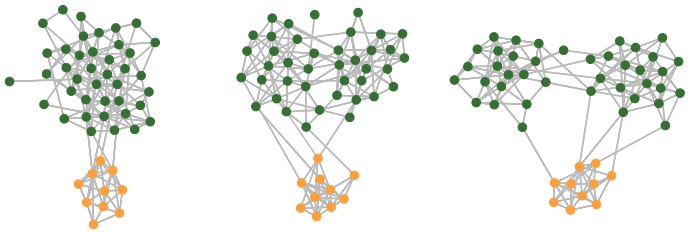
Since social network-based Sybil detection schemes use the structure of social networks to distinguish the Sybil nodes from the non-Sybil nodes, we begin by asking the following question: *Are there networks where it is hard to tell these two types of nodes apart?* In other words, could there be networks where the non-Sybil nodes look like Sybils or where it would be easy for Sybil nodes to masquerade as non-Sybils?

Intuitively, one would expect networks where the non-Sybil region is comprised of multiple, small, tightly-knit communities that are interconnected sparsely to be more vulnerable to Sybil attacks. In such networks, nodes within one community might mistake

non-Sybil nodes in another community for Sybils, due to limited connectivity between the communities. Furthermore, an attacker can easily disguise Sybil nodes as just another community in the network by establishing a small number of carefully targeted links to the community containing the trusted node. Next, we verify this intuition using experiments over synthetic and real-world social networks where the non-Sybil nodes have different community structures and the Sybil nodes use different attack strategies.

2.4.1 Impact of social network structure

We first examine the sensitivity of Sybil detection schemes to the structure of the non-Sybil region. As in Sections 2.2 and 2.3, we analyze synthetic networks and then show that the results from these simple cases apply to real-world networks as well.



(a) Initial (b) Intermediate (c) Final

Figure 2.10: Illustrations of the synthetic networks used in Section 2.4.1 (the actual networks are much larger). Non-Sybils are dark green and Sybils light orange. While the non-Sybil regions of (a), (b), and (c) show increasing amounts of community structure, all non-Sybil regions have the same number of nodes and links, and degree distribution.

We first generate a Bárabási-Albert random synthetic network [41] with 512 nodes and initial degree $m = 8$. This results in a random power-law network with approximately 3,900 links, and without any community structure. We then iteratively generate a series of networks by rewiring [40] five links in same manner as in Section 2.2 (resulting in a network), then rewiring five more links (resulting in another network), and so on, until only five links remain between the two communities of 256 nodes each (resulting in a final network). The output is a series of networks that all have the same number of nodes, number

of links, and degree distribution, but are increasing in the level of community structure that they exhibit. Figure 2.10 gives an illustration of the initial, intermediate, and final networks.

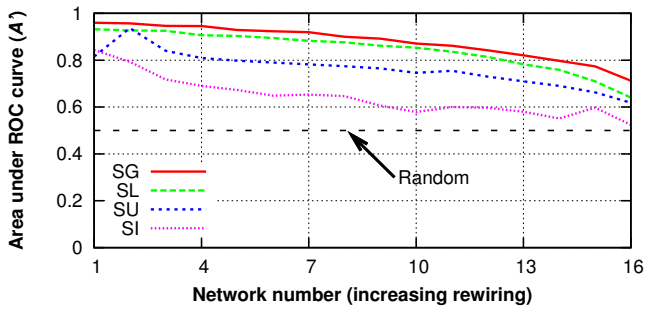


Figure 2.11: Accuracy of Sybil detection schemes on synthetic networks with increasing community structure induced by rewiring. With high levels of community structure, the accuracy of all schemes eventually falls to close to random.

We use this series of networks to evaluate how well Sybil detection schemes perform on networks with increasing amounts of community structure. To do so, we treat each of these networks as the non-Sybil region, and we randomly attach a Sybil region of 256 nodes using 40 links. We then evaluate how well the existing schemes are able to detect Sybils by using the A' metric. The result of this experiment for the final 16 networks are shown in Figure 2.11. It can clearly be seen that the Sybil detection schemes perform much better in the networks with less community structure than in those with more community structure. In fact, when there is a high level of community structure, the Sybil detection schemes perform close to what would be expected with a random ranking (indicated by a A' value of 0.5). Thus, the effectiveness of these schemes is very sensitive to the level of community structure present in the non-Sybil region of the network.

Next, we examine whether this observation holds in real-world networks. To do so, we collected a set of real-world networks that have varying levels of community structure, shown in Table 2.3. In order to measure the level of community structure present in the networks, we use the well-known metric *modularity* [159]. In brief, modularity ranges between -1 and 1, with 0 representing no more community structure than a random graph. Strongly positive values indicate significant community structure and strongly negative

Network	Nodes	Links	Modularity
Facebook undergrad [150]	1,208	43,043	0.278
Advogato [36]	5,264	43,027	0.318
Wikipedia votes [133]	7,066	100,736	0.350
URV email [114]	1,133	5,451	0.504
Astrophysicists [160]	14,845	119,652	0.621
Facebook grad [150]	514	3,313	0.644
High-energy physics [134]	8,638	24,806	0.690
Relativity [134]	4,158	13,422	0.790

Table 2.3: Size and modularity of the real-world datasets used in our evaluation. We assume all the graphs to be undirected and use the largest connected component.

values indicate less community structure than a random graph. As can be observed in the table, these eight networks have modularity value ranging from 0.28 to 0.79, indicating moderate to strong levels of community structure.

We conducted a similar experiment to the one above, treating these networks as the non-Sybil region, attaching a Sybil region, and evaluating the accuracy of Sybil detection. However, since these networks are of very different scales, we created a power-law Sybil region for each network with one-quarter the number of Sybils as there are non-Sybils, and attached these Sybil regions to the non-Sybils randomly with a number links equal to 5% of the links between non-Sybil nodes.

The results of this experiment are shown in Figure 2.12. We observe a clear trend: As the level of community structure increases, evidenced by increasing modularity, the performance

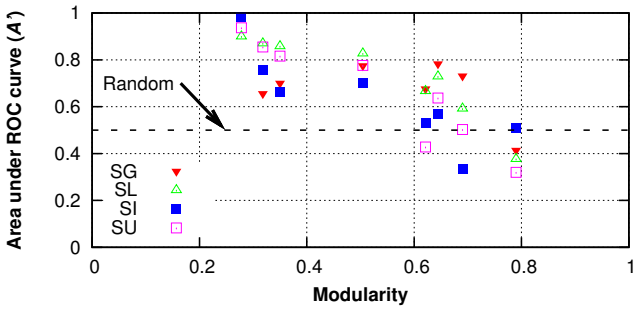


Figure 2.12: Accuracy of Sybil detection schemes on real-world networks from Table 2.3 with various levels of community structure. Significantly worse performance is observed as the level of community structure increases.

of the Sybil detection schemes falls close to random. In fact, a correlation coefficient of -0.81 is observed between the modularity value and the A' metric, demonstrating that increasing levels of community structure are strongly anti-correlated with the ability to distinguish Sybils. This poor accuracy also corresponds well with recent work [153] that has suggested that many real-world networks may not be as fast-mixing as was previously thought. Thus, as observed above for synthetic networks, Sybil defense schemes are extremely sensitive to the level of community structure present in real-world networks as well.

2.4.2 Resilience to targeted Sybil attacks

We now examine the sensitivity of Sybil detection schemes to Sybil attacks that leverage knowledge of the structure of the social network to establish links to a targeted subset of nodes in the network. Recall that all schemes assume that the Sybil nodes are allowed to create only a bounded number of links to non-Sybils. When evaluating the schemes, the authors of these schemes assume that the attacker establishes these links to random nodes in the network. We now explore how this one aspect of the attack model (random link placement to non-Sybils) can affect the performance of Sybil detection schemes by allowing the Sybils a level of control over where those links are placed. As before, we first examine the behavior using synthetic networks and then examine real-world networks.

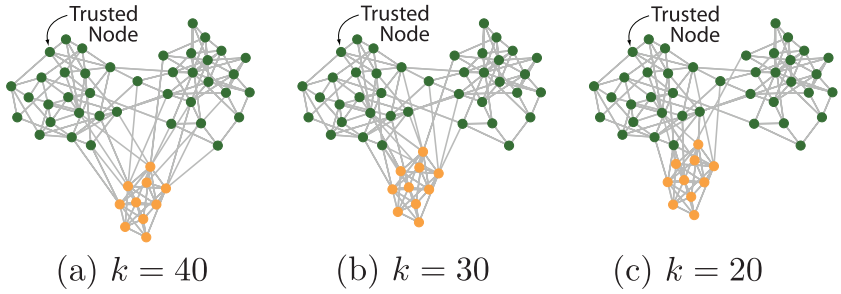


Figure 2.13: Illustrations of the synthetic networks used in Section 2.4.2 (the actual networks are much larger). Non-Sybils are dark green and Sybils light orange. With decreasing k, the Sybil nodes place their links closer to the trusted node.

To create the synthetic network, we use the methodology from Section 2.4.1, with rewiring done until only 40 links remain between the two communities of 256 nodes each. We then create a series of scenarios where we increasingly allow the Sybils more control over where their links to non-Sybils are placed. Specifically, instead of requiring the Sybil links to be placed randomly over the entire non-Sybil region, we allow the Sybils to place these links randomly among the k nodes closest to the trusted node, where closeness is defined by the ranking given by the community detection algorithm used in Section 2.3. In all cases, the number of Sybil-to-non-Sybil links remains the same. Thus, as k is reduced, the Sybils are allowed to target their links closer to the trusted node. We then calculate the accuracy of the Sybil detection schemes. An illustration of these networks is shown in Figure 2.13.

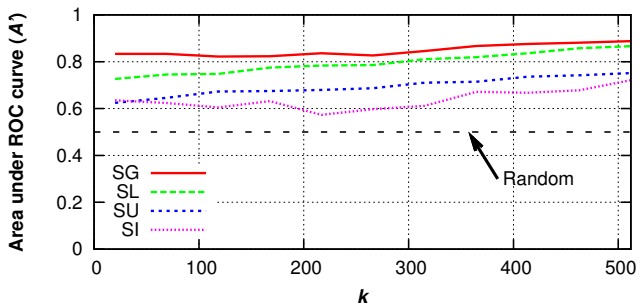


Figure 2.14: Accuracy of Sybil detection schemes on synthetic networks when Sybils are allowed to target their links among the closest k nodes to the trusted node. As the Sybils place their links closer (lower k), the accuracy of all schemes falls.

Figure 2.14 presents the results of this experiment. We see a decrease in accuracy as the Sybils are allowed to place their links closer to the trusted node. This is a result of the Sybil nodes being placed higher in the Sybil detection scheme's ranking, and therefore being less likely to be detected. From this simple experiment, it is clear that the performance of Sybil detection schemes is highly dependent on the attack model, depending (for example) on not just upon the number of links the attacker can form, but on how well those links can be targeted.

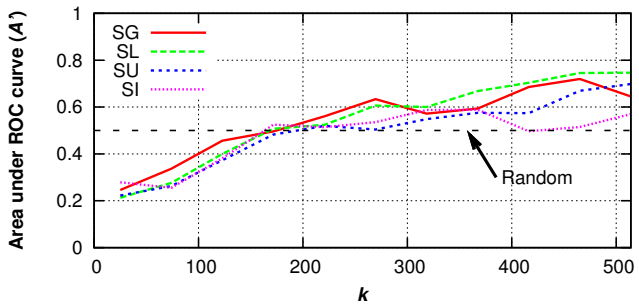


Figure 2.15: Accuracy of Sybil detection schemes on the Facebook network when Sybils are allowed to target their links among the closest k nodes to the trusted node. As the Sybils place their links closer, all schemes begin ranking Sybil nodes higher than non-Sybils (as evidenced by the A' below 0.5).

We then repeat the same experiment using the Facebook graduate student network. The results of this experiment are shown in Figure 2.15, and are even more striking than the previous experiment. As the attackers are allowed more control over link placement (i.e., as k is reduced), the accuracy first falls to no better than random, before dropping significantly below 0.5. This indicates that the Sybil detection schemes are ranking Sybils significantly higher than non-Sybils, meaning the schemes are admitting Sybils and blocking non-Sybils. The reason for this is the strong community structure present in the Facebook network combined with the stronger attack model: as the Sybils target their links more carefully, they appear as part of the trusted node's local community and are therefore more highly ranked.

2.4.3 Implications

In this section, we explored how the performance of Sybil detection schemes is affected by the structure of the social network and by the ability of the attacker to exploit the structure of the social network to launch targeted attacks. Based on our understanding of how Sybil detection schemes work, we hypothesized that networks with well-defined community structure would be more vulnerable to Sybil attacks. We verified our hypothesis by demonstrating that, as the non-Sybil region contains more significant community structure, the detection

accuracy of all schemes falls significantly and the schemes are vulnerable to targeted Sybil attacks.

Our analysis reveals fundamental limitations of existing Sybil detection schemes that arise out of their reliance on community structure in the network. Our list of limitations is by no means exhaustive; other vulnerabilities of relying on community detection exist. For example, a recent study has shown that identifying communities reliably in a wide range of real-world networks is a notoriously difficult task [135].

2.5 Concluding discussion

In this chapter, we have taken the first steps towards developing a deeper understanding of how the numerous proposed social network-based Sybil detection schemes work. We found that, despite their considerable differences, all Sybil detection schemes rely on identifying communities in the social network. Unfortunately, we also discovered that this reliance on community detection makes the schemes fundamentally vulnerable to Sybil attacks when operating over networks where the non-Sybil nodes form strong communities.

In light of these negative results, we look for alternative approaches to Sybil defense that could be deployed in practice. We discuss two ways to improve Sybil defenses moving forward. We present our discussion points as questions and answers.

Should Sybil defenses leverage more information? Given the inherent limitations of relying solely on the social network in order to defend against Sybils, an attractive way to improve on these schemes is to give Sybil defense schemes additional information. As a simple example, suppose a Sybil defense scheme were given a list of nodes, one in each of the different communities within the network, who were either known to be Sybils, or known to be non-Sybils. In this case, it is clear that this additional information could be used by community detection algorithms to accurately differentiate between communities containing Sybil and non-Sybil nodes. In fact, since our work was published [196], Cao et. al. [50]

proposed a Sybil detection scheme called SybilRank that leverages such extra information about the presence of known Sybils and non-Sybils in different network communities to more effectively detect Sybils. In contrast, current Sybil detection schemes are given only a single trusted node as input and consequently, they perform poorly.

As another example, recent work has suggested that *activity* between users may be a better predictor of the strength of the social link between them [202, 193]. These studies indicate that even in networks where users accept friend requests from arbitrary sources, users engage in shared activity (e.g., exchanging messages) with only a limited subset of their friends. Thus, having additional information about user activity could help weed out weak social connections, including links from Sybil nodes.

Should Sybil defenses move towards Sybil *tolerance?* Instead of explicitly identifying Sybils like SybilGuard, SybilLimit, and SybilInfer, a system could aim to instead just prevent Sybils from gaining access to extra privileges. SumUp, for example, attempts to limit the votes coming from Sybil nodes by limiting the effect of votes from potential Sybil regions. Instead of explicitly identifying nodes, the protocol seeks to limit their ability to disproportionally affect the resulting vote count. As a result, the system does not try to prevent users from creating multiple identities, but rather, ensures that by doing so, they are unable to gain any additional privileges. This idea leads us to our next chapter that focuses on social network-based Sybil tolerance schemes.

CHAPTER 3

Towards social network-based Sybil tolerance

In this chapter, we present an alternate approach to defend against Sybils by leveraging the social network. Instead of trying to explicitly label identities as Sybil or non-Sybil, these schemes are designed to limit the impact that a malicious user can have on others, regardless of the number of identities the malicious user possesses. We refer to these schemes as *Sybil tolerance* [194]. Similar to Sybil detection schemes, these schemes are also based on the assumption that, although an attacker can create an arbitrary number of Sybil identities, she cannot establish an arbitrarily large number of social connections to non-Sybil identities. However, Sybil tolerance schemes do not make the assumption that the non-Sybil region of the network is fast-mixing [151]. Instead, they require more information about the system to which they are applied: In addition to the social network, these schemes also take as input the interactions between users. By doing so, they are able to allow or deny individual interactions, and reason about the impact (in terms of interactions) that identities have on one another.

The result is that the guarantees of Sybil tolerance are expressed in terms of interactions that are allowed. To compare, Sybil detection schemes reason only about identities (i.e., they reason about identities being *admitted*, and express their guarantees in terms of the number of Sybil identities admitted), while Sybil tolerance schemes reason about interactions (i.e., they decide whether certain interactions are allowed or denied). Thus, in a Sybil tolerance

scheme, a certain pair of identities may be allowed to participate in certain interactions and not others, and may be allowed to interact at certain times and not others (all depending on the state of the system). These schemes have been shown to be effective in applications including reputation systems [71], spam protection [149], online auctions [165], and content rating systems [185].

A few examples of social network-based Sybil tolerant systems already exist in the literature [149, 185, 165]. However, compared to Sybil detection, Sybil tolerant systems are less well-studied. Consequently, their design patterns and their properties are poorly understood. Existing Sybil tolerant systems are application-specific solutions, and it is unclear how to leverage social networks to integrate Sybil tolerance into other social computing systems.

In this chapter, we make the following three contributions: First, we propose a general methodology for designing social network-based Sybil tolerant systems. Our methodology is based on *credit networks* [64, 104], a concept we borrow from the electronic commerce community. Credit networks were first introduced in the electronic commerce community in order to build transitive trust protocols in an environment where there is only pairwise trust between accounts and there are no central trusted entities. We consider a social computing system where users perform pairwise transactions (e.g., sending a message, purchasing an item, casting a vote). In such a system, we form a credit network by assigning *credits* to the social network links, and then allowing actions only if *paths with sufficient credit* exist between the source and destination of an action. We show that credit network-based Sybil tolerant systems built from social networks are naturally tolerant to Sybil attacks. We ensure that the number of transactions that a (human) user can initiate is independent of the number of identities she possesses. Such a guarantee removes the creation of multiple accounts as an attack vector, thereby making the application tolerant of Sybils. We further discuss the key challenges associated with building such credit network-based Sybil tolerance schemes.

Second, we demonstrate that despite their differences, all proposed Sybil tolerance systems work by conducting payments over credit networks. Effectively, these schemes are using credit networks as a basis for Sybil tolerance. Unfortunately, these schemes do not scale well to large social networks. Finding routes for credit payments can be reduced to determining the maximum flow [98] between nodes in the network; doing this over large graphs is known to be expensive [107], and existing techniques for pre-calculating [108] the maximum flow are not directly applicable since the credit network is constantly changing. This serves as a practical deployment barrier, and to the best of our knowledge, none of these Sybil tolerance schemes have been deployed in a real-world system.

Third, we address this situation and scale Sybil tolerance schemes to extremely large graphs. We build Canal, a system that can efficiently approximate credit payments over large, dynamic networks. Canal trades accuracy for speed; we demonstrate that Canal's approximation rarely impacts users and does not change the Sybil tolerance properties of the application or benefit malicious users. We show that Canal can be directly plugged into existing Sybil tolerance schemes, and would reduce the credit payment latency from multiple seconds to a few hundred microseconds.

In more detail, Canal uses a novel landmark routing-based algorithm, routing credit payments via landmark nodes [187]. Canal consists of two components: a set of *universe creator* processes, which continually select new landmarks, and a set of *path stitcher* processes, which continually process incoming credit payment requests. Since the credit network is constantly changing (due to credit movements, as well as new identities and social links), Canal continually calculates new landmarks in parallel with making flow calculations. We design Canal to naturally take advantage of multiple cores and machines, enabling Canal to run over social networks that cannot be stored on a single machine.

We evaluate Canal on real-world networks at scale. We first demonstrate that Canal's approximation provides a dramatic speedup in the processing of credit payments, enabling Canal to be run in an online fashion. We then show that the approximation that Canal

uses rarely impacts users, and that users eventually receive the same total available credit in Canal as they would in an exact system. Finally, we re-run the experimental setup of two previously proposed Sybil tolerance schemes, and demonstrate that using Canal would provide up to a 2,329-fold speedup in runtime while maintaining over 94% accuracy. This shows that existing schemes can naturally leverage Canal, and that Canal enables new schemes to be designed to inherit the benefits of credit networks.

3.1 Background and related work

In this section, we give a brief overview of the prior work on social network-based Sybil defenses, with the goal of placing the contributions of this chapter into context. We divide the related work on social network-based Sybil defense into two classes, discussed in the sections below: Sybil detection, and Sybil tolerance.

3.1.1 Sybil detection

We presented *Sybil detection* approaches in Chapter 2. In these schemes, researchers have explored allowing Sybil identities to be created but later detecting the identities and preventing them from interacting with other users (e.g., banning those identities) [176]. All schemes focus on analyzing the structure of the social network as a mechanism for Sybil detection [210, 209, 65, 184, 166, 138]. To identify Sybils, all social network-based Sybil detection schemes make two common assumptions [208]:

1. Although an attacker can create an arbitrary number of Sybil identities in the social network, she cannot establish an arbitrary number of social connections to non-Sybil identities.

2. The non-Sybil region of the network is densely connected (or fast-mixing [151]), meaning random walks in the non-Sybil region quickly reach a stationary distribution.

The first assumption concerns how the Sybil and non-Sybil identities are connected and is necessary in order for the schemes to be able to leverage the social network; if it were not assumed, the attacker could establish social network links at will. While this assumption may not hold on all online social networks, recent work suggests that there are social networks where this assumption holds true [155]. The second assumption concerns the internal structure of the non-Sybil region and is necessary for these schemes to locate the boundary between the non-Sybil region and the Sybil region. If the second assumption did not hold (implying small cuts existed within the non-Sybil region), the honest identities on either sides of cuts are likely to be blocked from interacting with each other [196].

3.1.2 Sybil tolerance

Sybil tolerance schemes [194] take an alternate approach to defend against Sybils. Sybil tolerance schemes make the same assumption 1 from Section 3.1.1, but avoid making assumption 2. However, these schemes require information about pairwise interactions between users in addition to the knowledge of the social network structure.

We now provide a brief overview of three example Sybil tolerance schemes. It is important to note that other Sybil tolerance schemes exist [71, 154, 123], but we only discuss the three below for brevity.

Ostra [149] is targeted at countering unwanted communication (i.e., spam). Ostra assumes the existence of a social network, and assigns credit values to the links. When a user wishes to send a message to another user, Ostra locates a path with available credit from the source to the destination. If such a path is found, credit is “paid" from each user to the next along the path, and the credit is refunded if the message is not marked as spam. If no path can be found, the message is blocked from being sent.

SumUp [185] (also discussed in Chapter 2) is designed to prevent users with multiple identities from manipulating object ratings in content sharing systems like Digg. SumUp

assumes the existence of a social network and selects a trusted vote collector. SumUp then assigns weights to the links around the vote collector by handing out “tokens" (causing the links around the vote collector to be more highly weighted; links farther away are assigned weight 1). Finally, to vote, each voting identity must find a path with credit between himself and the vote collector and consume a credit along that path; if no such path can be found, the vote is discarded.

Bazaar [165] provides stronger user reputations in online marketplaces like eBay. To do so, Bazaar creates a transaction network (akin to a social network) by linking pairs of identities that have successfully completed a transaction; the weight of each link is the dollar value of the transaction. When a new transaction is about to take place, Bazaar compares the value of the new transaction to the max flow between the buyer and seller. If sufficient flow is found, credit totaling the value of the transaction is removed between the buyer and seller, and is added back if the transaction is later reported to not be fraudulent. Otherwise, if sufficient flow is not found, the new transaction is denied.

3.2 Sybil tolerance and credit networks

We now show that social network-based Sybil tolerance schemes can be implemented using *credit networks*. We first give a brief overview of credit networks before describing the Sybil tolerant nature of credit networks.

3.2.1 Credit networks

Credit networks [64, 104] were first introduced in the electronic commerce community in order to build transitive trust protocols in an environment where there is only pairwise trust between accounts and there are no central trusted entities. In a credit network, identities (nodes) trust each other by offering pairwise credit (links) up to a certain limit. Nodes can use the credit to pay for services they receive from each other. The credit network could also

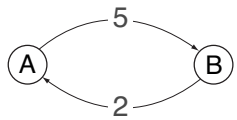


Figure 3.1: Simple credit network between two nodes A and B, with credit available c_{ab} and c_{ba} shown. In this example, A has 5 credits available from B, and B has 2 credits available from A.

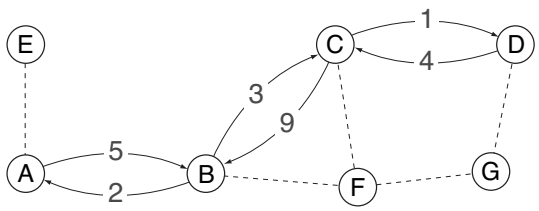


Figure 3.2: More complex credit network, with credit available (c_{ij}) shown for each link. In this example, A can pay 1 credit to D along the path $A \to B \to C \to D$. After paying the credit, the values on these links would be 4, 2, and 0, respectively. Note that, for simplicity, the links not on this path are only shown as dashed lines.

be used as a payment infrastructure between nodes that do not directly extend credit to each other. Nodes can route credit payments to a remote node via network paths that traverse over links where credits are available (see Figures 3.1 and 3.2).

Formally, a *credit network* is a directed graph $G = (V, E)$ where V is the set of nodes and E is the set of labeled edges. Each directed edge $(a, b) \in E$ is labeled with a dynamic scalar value c_{ab}, called the *available credit*, and is initialized to C_{ab}. Intuitively, C_{ab} represents the initial credit allocation that b gives to a, and c_{ab} represents the amount of unconsumed credit that b has extended to a. Note that $c_{ab} \geq 0$ at all times.

Payments between two nodes in a credit network are contingent upon the availability of credit along network paths connecting the nodes. If a node a wishes to pay node b a total of c credits, then a path

$$a \to u_1 \to ... \to u_n \to b$$

(which could just be $a \to b$) must exist where c credits are available on each (i, j) link (i.e., $c_{ij} \geq c$). If so, the credit available on each directed edge c_{ij} on the path from a to b is

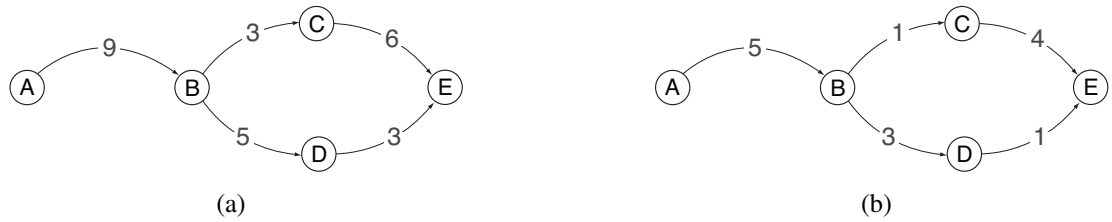


Figure 3.3: The (a) initial and (b) final state of the credit network with a credit payment along multiple paths. A pays 4 credits to E: 2 credits are paid along the path $A \to B \to C \to E$ and 2 credits are paid along the path $A \to B \to D \to E$.

decreased by c. As a result of this action, each node "pays" c credits to its successor on the path to b.

It is not necessary to find a single path with available credit along each edge; instead, the payment could be split across multiple paths. For example, consider the network shown in Figure 3.3. In this scenario, node A could pay 4 credits to node E by paying 2 credits along $A \to B \to C \to E$ and 2 credits along $A \to B \to D \to E$.

3.2.2 Credit networks from social networks

One can build a credit network from a social network as follows: For each identity in the social network, we generate a node in the credit network. For each edge between a pair of identities in the social network, we generate an edge in the credit network between nodes corresponding to the users. Undirected edges in the social network (e.g., Facebook friend links) are replaced by two directed edges, one in each direction, between the nodes adjacent to the edges. Because social networks are known to be richly connected [148, 37], credit networks inherit the rich connectivity they require for liquidity [64].

Further, each directed edge, (a, b), is assigned an initial credit allocation C_{ab} by the destination node b. The system must exercise care when assigning credit allocations. For instance, when a new social link is created, the requesting node should be required to grant the accepting node some initial credit but not vice-versa, to prevent an attacker from obtaining credit by initiating social links.

3.2.3 Sybil tolerant nature of available credit

Credit networks have been shown to be naturally tolerant to Sybil attacks [172]. In brief, we assume that an attacker is allowed to create as many identities as she wishes and manipulate the available credit on links between identities she owns. However, the attacker is able to establish only a limited number of links to non-malicious users (by assumption 1 in Section 3.1.1), and she cannot manipulate the credit available to her on these links.

As shown in Figure 3.4, the total amount of available credit to the malicious user is the sum of the available credit on her links to other users. An attacker with an arbitrary number of Sybil identities has exactly the same available credit as the attacker with just one identity; in this case, the relevant set of edges is the cut between the subgraph consisting of the attacker's Sybil identities and the rest of the network. Any available credit on edges between the attacker's Sybil identities does not matter, because it does not enable additional payments to legitimate nodes.

Thus, available credit in a credit network is resilient to Sybil attacks.

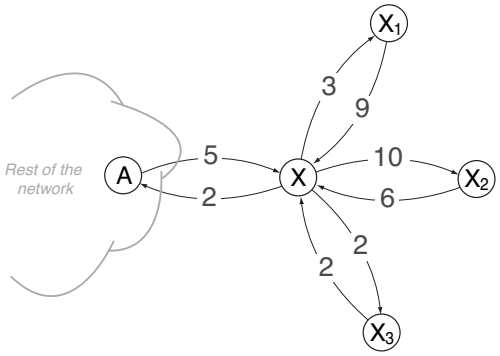


Figure 3.4: Credit networks leading to Sybil tolerance. User X can create any number of identities (X_1, X_2, X_3) and arbitrarily assign the credit available between them. However, if X wishes to pay credits from any of these identities to another identity in the rest of the network, the credits must be debited from X's single valid link to A. Thus, the multiple identities do not enable any additional available credit with nodes in the rest of the network.

3.3 Credit networks in existing systems

We now discuss how the three example schemes discussed in Section 3.1.2 all work by essentially performing payments over credit networks.

First, we note that each of the schemes internally uses a credit network: In Ostra and SumUp, the credit network is based on an externally provided social network, and in Bazaar, the credit network (called the *risk network*) is constructed from the feedback on prior transactions.

Second, we observe that each scheme assigns available credit to links: In Ostra, the initial available credit is statically defined by the system operator, in SumUp, the credit is assigned by a token distribution mechanism, and in Bazaar, the credit is increased after each successful transaction.

Third, we observe that these schemes work by *paying credits* along paths between identities. In Ostra, a sender must first pay a receiver one credit before sending a message; if the sender is out of credit, the message is not delivered. In Ostra, when a sender pays a credit to a receiver, a credit is debited from the sender–receiver path and, at the same time, *added* to the reverse path. Doing so allows Ostra to ensure liveness (as there is always credit available in the system). Similarly, in SumUp, each voter must pay one credit to the vote collector; if no path exists between the voter and vote collector with available credit, the vote is not counted. Finally, in Bazaar, when a transaction is about to occur, the system insists that the buyer pay the seller a number of credits corresponding to the new transaction value; if the sufficient available credit does not exist, the transaction is blocked. In Bazaar, this credit payment is undone if the transaction turns out not to be fraudulent.

3.4 Challenges building credit network-based Sybil tolerance systems

We now discuss the key challenges associated with building credit network-based Sybil tolerance systems. This includes two main challenges: fundamental credit network design challenges and practical implementation/deployment challenges.

3.4.1 Credit network design challenges

The credit network plays a fundamental role in the operation of Sybil tolerance schemes. Thus, a system designer wishing to apply Sybil tolerance in a given application must make careful choices concerning the initialization of starting credits on links, the adjustments to credits after each transaction, and the replenishment of credits over time. The key goals are to maintain *liquidity in the credit network* such that (1) most attacker transactions are disallowed, (2) most legitimate transactions are allowed, and (3) the credit network does not introduce any denial-of-service attacks on legitimate users. The challenge is to design a credit-network based mechanism that encourages legitimate transactions and discriminates against unwanted transactions.

We discuss these goals in more detail using Ostra as an example. Recall that in Ostra's credit payment mechanism, a decrease in available credit on an edge on the path from a to b is accompanied by an increase in available credit on any edge from b to a. The decrease in available credit along an edge from a to b reflects the difference in number of mutual favors or service asked by a and b. Such a payment mechanism bounds the imbalance or difference in the number of service requests between the connected nodes.

Bounding undesirable transactions As discussed earlier in Section 3.2.3, credit networks built from social networks are Sybil tolerant by nature. In the attack topology shown in Figure 3.5, the imbalance in transactions between the spammers and legitimate users (i.e.,

the spam in our messaging system) is always bounded by the aggregate credit (the sum of the credit balances on the links) available on the edge cut separating spammers from legitimate users. This is true regardless of the number of Sybil identities the spammers use or the credit balances on the links between the spammers' identities. Thus, the credit network naturally bounds the number of spam transactions, regardless of the number of identities the attacker possesses.

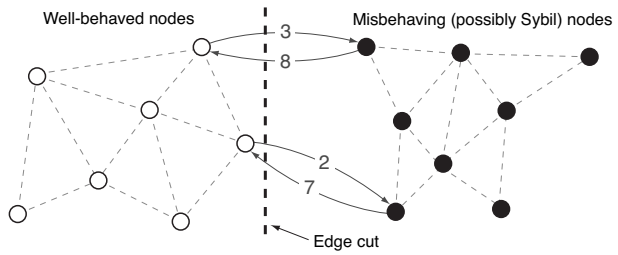


Figure 3.5: Edge cut between well-behaved nodes (hollow) and misbehaving nodes (solid). The total credit available to the misbehaving nodes is 5 (3+2), regardless of the number of Sybil identities created. Note that the links that are not along the edge cut are shown as dashed lines, for simplicity.

Allowing legitimate transactions The system designer must also ensure that the chosen mechanism does not block legitimate transactions in the common case. So next, we focus on the case when all nodes in our messaging system are legitimate. Consider an edge cut that divides legitimate users into two groups, as shown in Figure 3.6. Our credit adjustment mechanism would bound the credit imbalance between the two groups to the credit each of the groups make available to the other. If the identities in one group are interested in sending a disproportionately large number of messages to the identities in the other group, the credit along the edge cut could be exhausted, preventing further transactions. This is essentially a *liquidity* problem, where a subset of the legitimate nodes have insufficient liquidity with another subset.

Thus, in the long-term, any subset of legitimate nodes must receive messages from the rest of the legitimate nodes as often as it sends messages to those nodes.[1] The mechanisms

[1]Short-term imbalances can be absorbed by setting appropriate initial credit allocations.

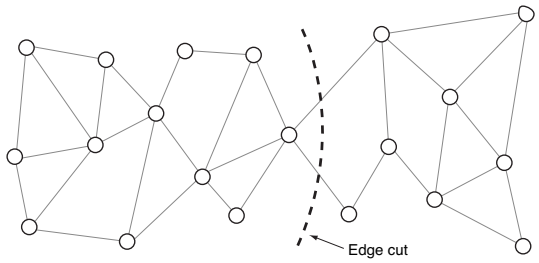


Figure 3.6: Edge cut internal to the well-behaved nodes.

should be chosen so that the statistics of a legitimate workload distribution would ensure an approximate long-term trade balance. In another work, we designed a Sybil tolerance scheme to limit large scaling web crawling in social networks by leveraging the long-term trade balance in real-world legitimate social network interactions (users browsing each other's profile) [154]. Also, the use of techniques like credit replenishment (where credits are periodically readjusted by the system) can be used to maintain liquidity [149, 154].

Vulnerability to attacks on network liquidity Finally, the system designer must ensure that the credit network mechanism does not introduce new vulnerabilities. For example, can a few attacker nodes exhaust the credit along an edge cut separating legitimate identities, thereby preventing the legitimate identities from interacting with each other? For example, consider a small cut A through the network where there are both attacker and legitimate identities on either side. If the attackers have, in aggregate, more credit with the legitimate identities than exists along cut A, it is feasible that the attackers could exhaust the credit along A (e.g., by sending messages to each other, affecting the credit values on A).

There are social networks that are sufficiently well connected that the min-cut between any pair of nodes tends to be adjacent to either of the nodes [149]. It follows that a single misbehaving node will run out of credit before the credit on any other cut in the network is exhausted (see Figure 3.7). Further, a group of Sybils controlled by an attacker will tend to have a small cut to the rest of the network (because the attacker is unable to create an arbitrary number of links to other real users). Therefore, a group of Sybils is also likely to

run out of credit before the before the group can exhaust the credit on any larger cut in the network.

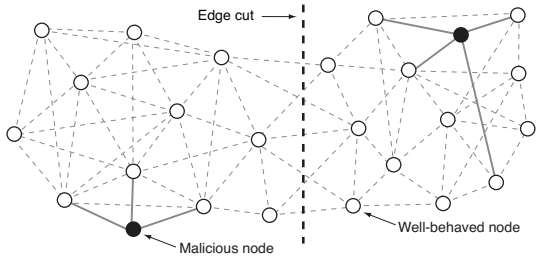


Figure 3.7: Diagram of the resilience of credit networks to credit exhaustion attacks by malicious nodes (shown as filled nodes). In some real-world social networks, the min cut between nodes occurs at the nodes themselves, rather than in the middle of the network, preventing malicious nodes from exhausting credit between well-behaved nodes

3.4.2 Deployment challenges

Credit network-based Sybil tolerance schemes face scalability challenges when applied to large social networks. First, we observe that performing a credit payment involves searching for one or more paths with available credit between two nodes; this is essentially the maximum flow problem [98], which is known to be a computationally expensive operation. The most efficient algorithms for the maximum flow problem run in $O(V^3)$ [107] or $O(V^2 \log(E))$ [70] time. Second, techniques that pre-calculate the all-pairs maximum flow (e.g., Gomory-Hu trees [108]) cannot be applied to Sybil tolerance schemes, as these techniques assume a static network and impose a large, upfront pre-calculation cost of $|V| - 1$ maximum flow computations. Credit networks are constantly changing due to credit manipulations as well as new users and links; performing $|V| - 1$ maximum flow computations for every change in the credit network is impractical. Additionally, algorithms [117] that dynamically maintain a Gomory-Hu tree when the edge capacities increase or decrease often end up being expensive as well, requiring several maximum flow computations for each edge capacity update.

For example, Bazaar can take over 6 seconds [165] to determine whether sufficient flow exists over a network with 3.3 million links. Given that Bazaar is intended to be run in an online fashion, Bazaar would require an average of 6 seconds of CPU time for every bid on a marketplace like eBay. This represents substantial computational resource investments. With ever-larger and denser networks being created, it is unsurprising that—to the best of our knowledge—none of the existing Sybil tolerance schemes have been deployed in a real-world system.

3.4.3 Rest of the chapter

In the rest of the chapter, we focus on the key deployment challenge faced by designers of Sybil tolerance schemes—scaling up Sybil tolerance schemes to work on very large social networks. To this end, we present the design and implementation of Canal, a system that can efficiently process credit payments over large credit networks.

3.5 Canal design

We now detail the design of Canal, first giving a high-level overview of the model and goals of Canal before detailing the internal design.

3.5.1 Model and goals

Canal is designed to run alongside an existing Sybil tolerance scheme, providing two services: (a) maintaining the state of the credit network and (b) conducting credit payments. Canal is built to provide these services in a much faster manner than current implementations. We assume that Canal is run by the same organization that runs the Sybil tolerance scheme (alleviating concerns about Canal having access to potentially private social network data).

In order for existing Sybil tolerance systems like Bazaar and SumUp to take advantage of Canal, they need to only allow Canal to store the credit network and replace any internal

logic for conducting payments with calls into Canal. To avoid having to rebuild existing systems from scratch, Canal exports an API which can be easily used by existing Sybil tolerant applications. The API includes methods to initialize and add links to the credit network, but we are primarily concerned with the method

```
boolean payment(a,b,c)
```

that attempts a payment of c credits from identity a to identity b, and returns whether or not the payment could be made.

Canal responds to payment requests using an approximation that only considers a subset of the paths that exist when handling payments. As a result, Canal may not be able to find paths with available credit between a pair of users even though such paths exist. However, Canal will never find paths that do not exist or paths that do exist but do not have any available credit. Thus, Canal can suffer from *false negatives* (i.e., a payment is denied even though paths with available credit exist) but does not suffer from *false positives* (i.e., a payment is allowed even though sufficient credit is not actually available).

3.5.2 Design challenges

In order for Canal to be used in a real-world application, it has to overcome several challenges:

- *Latency:* Sybil tolerance schemes make user-visible decisions based on whether credit payments can be made. Thus, they will be practically deployable only if the Canal processing time is very fast (preferably in the order of a few milliseconds).

- *Efficiency:* Sybil tolerance schemes often have to process large numbers of payments in a short period of time; Canal must be deployable with reasonable computational resources.

- *Scalability:* Sybil tolerance schemes are designed to be run on very large social networks. Canal should support credit networks with hundreds of millions of links or more.

- *Accuracy:* Canal trades off accuracy for speed. The error introduced should not impact the Sybil tolerance guarantees, and should rarely impact users.

- *Dynamicity:* The credit network is constantly changing, due to payments being processed and changes to the social network. Canal should be able to support such a rapidly changing credit network.

3.5.3 Using landmark routing

Canal speeds up payments using a landmark routing-based technique. Historically, landmark routing [187] has enabled paths to be found between any pair of nodes via certain specific nodes (called landmarks). To do so, each node determines its path to the landmark; to route between a pair of non-landmark nodes, we need only have each route to the landmark. This is effectively *stitching* a path together out of two paths, and the stitched path may be longer than the shortest path (this is particularly likely when the landmark is located far away from the two nodes).

Canal's selection of landmark nodes is driven by three concerns, discussed in detail in the subsections below: First, we wish to be able to find short paths between nodes, but we are not required to use the absolute shortest path (Sybil tolerance systems are designed so that *any* payment path will do, but shorter paths often result in greater efficiency). Second, landmark routing is not typically designed for dynamic graphs (the paths to the landmark are generally treated as static; if the credit network is changing, Canal needs to update the paths to the landmarks). Third, we may need to conduct payments that require multiple paths (the credit available on any single path may not be sufficient).

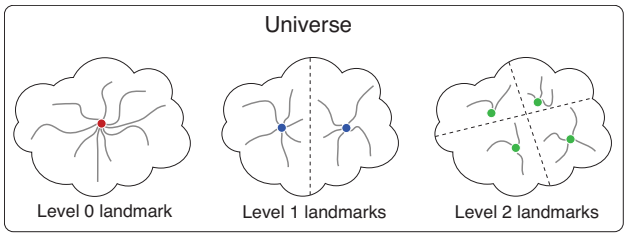


Figure 3.8: Diagram of a 2-level landmark universe, consisting of multiple levels of landmarks. Each level i has 2^i landmarks. Paths can be found using landmark routing; all nodes share a level 0 landmark, and closer nodes share landmarks at multiple levels (resulting in shorter paths). The landmarks at lower levels induce partitions on the network (indicated by dashed lines).

3.5.3.1 Finding short paths

Recent work has designed landmark routing techniques for accurately finding shortest paths in large networks [113]; we leverage this existing work to efficiently find short paths. Instead of using a single landmark, Canal uses a *landmark universe* with multiple *levels*. At each level i, there are 2^i landmarks selected randomly; each node finds a path to the closest landmark at each level. Thus, if there are k levels, there are a total of $2^{k+1} - 1$ landmarks, and each node has paths to k of these landmarks (one at each level). Furthermore, we refer to a universe with k levels as a k-level landmark universe. A diagram of a landmark universe is shown in Figure 3.8.

Using a landmark universe enables Canal to find short paths. To see why, first consider a payment request between two nodes who are on opposite sides of the network (i.e., two nodes who have a relatively large shortest path length). For this pair of nodes, the only landmark they are likely to share is the level 0 landmark[2] (since they are far away, they are likely to have paths to two *different* level 1 landmarks, and two different level 2 landmarks, etc). Now consider the case of a payment request between two nodes who are close in the network. For this pair of nodes, there is likely to be a number of landmarks shared between them (since they are close in the network). By stitching a path between these nodes via

[2]Note that any pair of nodes is guaranteed to share at least one landmark in a given universe (the level 0 landmark), although the stitched path via that landmark is not guaranteed to have credit.

one of the higher-level landmarks, we are likely to find a short path. A full explanation is available in [66, 113].

It is important to note that Canal's correctness guarantees are not affected if a Sybil node is selected as a landmark. On the other hand, Sybil landmarks may affect the liveness of paths stitched via that landmark, as links near the landmark may not have credit. As a result, a Sybil landmark may not be useful for routing credit payments. Since Canal typically uses hundreds of landmarks at any time, Sybil landmarks rarely impact Canal's ability to route credit payments.

In Section 3.6, we show how the deployer of Canal can select the level k of each universe. Higher values of k allow shorter paths to be located, but introduce an exponentially increasing number of landmarks (and corresponding overhead). In practice, setting k to 5 works well on the social networks we use to evaluate Canal.

3.5.3.2 Handling dynamic credit networks

We observe that, due to the rapidly changing nature of the credit network, any existing landmark data may quickly become stale. For example, as new links are introduced into the credit network, paths may exist that are not reflected in the landmark universe. Similarly, as credit payments are processed, paths near lower-level landmarks are likely to become congested and may run out of available credit (since a disproportionate number of credit payments will be routed via these landmarks); this would prevent Canal from finding available credit that may exist via other potential landmarks.

Canal addresses this issue by continually constructing landmark universes as it is running, replacing an old universe whenever a new one is created. This serves two purposes. First, continually constructing universes enables Canal to incorporate changes in the credit network into the landmark path data. Second, continually constructing universes ensures that any node is only a landmark for a short period of time, reducing the likelihood that the paths around the landmark would become congested with credit payments.

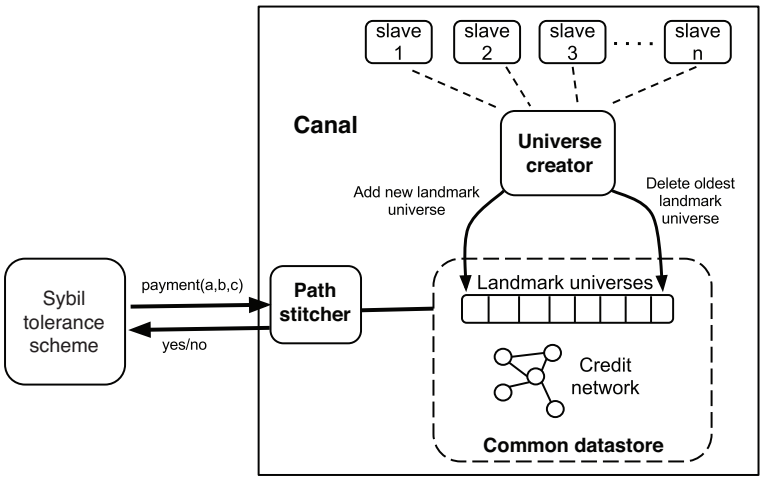


Figure 3.9: Canal system design.

3.5.3.3 Finding multiple paths

We noted above that Canal may need to conduct payments that require multiple paths (i.e., in the example in Figure 3.3, suppose A wishes to pay 5 credits to E). In particular, we would like to be able to locate *disjoint* paths, as this increases our likelihood of finding the necessary available credit.

Canal addresses this issue by keeping a queue of recent landmark universes available. As new landmark universes are generated, they are added to the end of the queue and the oldest existing landmark universe is discarded. Keeping a set of universes available enables Canal to find paths between pairs of nodes via the landmarks that exist in different universes. By configuring the number of landmark universes that are stored in the queue, Canal can control the maximum number of paths that can be used at once for a credit payment.

3.5.4 Canal components

Figure 3.9 gives a high level view of Canal system design. There are three main components in Canal: a common datastore, *universe creator* processes, and *path stitcher* processes. The common datastore serves as a location to store the landmark universes and the credit network. The universe creator processes continually generate new landmark universes as

described above, and the path stitcher processes respond to payment requests by the Sybil tolerance scheme. In the rest of this section, we will discuss in detail the design of each component.

3.5.4.1 Common datastore

The common datastore serves as the repository for the state of the credit network as well as the landmark universes. The credit network is stored as a hash table of links, with each link stored with its current available credit. Because multiple processes will be manipulating the values on the credit network, each link also contains a shared/exclusive lock that processes must obtain before reading/writing the credit value. When a new link is added to the credit network, its lock and credit available are first initialized before it is inserted into the credit network hash table.

The landmark universes are stored using a linked list, with a global pointer to the head of the landmark universe list and each landmark universe pointing to the next. Since multiple processes will be scanning the landmark universe list, the pointers are also protected with read/write locks which processes obtain before following or changing a pointer.

The landmark universes themselves are represented as a series of *landmark maps* with one landmark map for each level in the landmark universe. Each landmark map contains tuples

```
(node, landmark, next_hop)
```

with one entry for each node in the network. The `landmark` represents the given node's landmark at this level, and the `next_hop` represents the node's next hop towards this landmark. By recursively following the `next_hop`s, each node can reconstruct its path to the landmark. Thus, in a k-level landmark universe, there are k landmark maps, each with an entry for all nodes in the network. Thus, each landmark universe requires $O(n \cdot k)$ space, where n is the number of nodes in the credit network.

3.5.4.2 Universe creator processes

We now describe the design of the universe creator processes, which construct new landmark universes. Assume that Canal is configured to construct k-level landmark universes. The universe creator processes all continually construct landmark universes using the following approach, taken from [113].

1. Randomly select k random node sets of sizes $2^0, 2^1, 2^2, 2^3, \ldots 2^k$ respectively, from the network. Let the selected sets be denoted by $V_0, V_1, V_2, \ldots V_k$. These sets contain the new landmarks at each level.

2. For each set V_i, and every node $u \in V$, calculate the shortest path from u to each of the landmark nodes in each set V_i. This is done by having the processes perform BFSs from each landmark in V_i.

3. Finally, using the BFSs, construct the landmark map for level V_i by select the closest landmark node in V_i and the next hop for all nodes.

In Canal, we speed up this universe creator process using three techniques: First, Canal exploits the fact that conducting the BFSs from the new landmarks can be parallelized. We configure the BFSs to be conducted in parallel by a set of slave threads. Second, to make sure that we only find paths with available credit, we design the BFS algorithm to only consider edges with available credit. This allows newly constructed landmark universes to "route around" links which have no available credit (and cannot be used for payments).

Third, the process of selecting the closest landmark at a given level for all nodes (step 3 above) can be broken down into a series of *merges* that can easily be parallelized as well.[3] Let us consider the 2^i BFSs that are conducted when constructing the landmark map for level V_i. Note that these BFSs are completed at different times by different processes. The

[3]Our current implementation does not support the parallel BFS merge feature. We plan to incorporate this in the future.

landmark map is constructed by taking the first BFS and creating an entry for every node in the landmark map pointing to the landmark at the root of the BFS. Then, as subsequent BFSs complete, they are merged into the landmark map by scanning over the existing landmark map, and changing any tuples where the node is closer to the new landmark than to any landmark previously merged. In fact, multiple landmark maps can be merged together in the same manner, allowing all of the processes to contribute to constructing the landmark map.

Once the new landmark universe is constructed, it is added on to the end of landmark universe list in the common datastore. At the same time, the oldest landmark universe is removed from the front of the list and discarded. This ensures that landmarks are only "active" for a short period of time, reducing the likelihood that they will become hotspots in the network (Section 3.6 shows this happens rarely).

3.5.4.3 Path stitcher processes

Finally, we describe the design of the path stitcher processes, which respond to `payment` requests from the Sybil tolerance scheme. Let us suppose that a path stitcher process has received a request to pay c credits from node a to b. At a high level, the path stitcher process walks down the landmark universe list, looking for paths with available credit between a and b using the landmarks in each universe. As soon as the path stitcher process has found paths with a total of c available credits, it returns a successful result. Otherwise, if the path stitcher process reaches the end of the universe list without finding a total of c credits, it returns an unsuccessful result.

To find paths with available credit in a single k-level landmark universe, the path stitcher process executes the following algorithm:

1. Scan the k landmark maps and collect the set of common landmarks between a and b. Note that there is guaranteed to be at least one common landmark (at level 0).

2. For each shared landmark, use the `next_hop` in the landmark map to "stitch" together a path via the landmark.

3. Refine the path by (a) eliminating any cycles and (b) performing path *short-cutting*. To perform short-cutting, we traverse the path up to the landmark node and see if there is a link from any of these nodes to a node lying in the path after the landmark node. If so, we short-circuit the path by using that link to create a shorter path between a and b.

This process results in up to L paths between a and b, where L is the number of common landmarks in this universe.[4]

Next, the path stitcher process pays as much credit as possible along each path. For each path, the path stitcher process walks the path, obtaining the lock on each link of the path, temporarily lowering the credit available to 0, and then releasing the lock. Once the end of the path has been reached, the path stitcher calculates the maximum credit available on the entire path (determined by the link with the minimal credit available); let this be C. Then, the path stitcher process walks the path once more, locking each link and resetting the credit available to be its previous value minus C. This effects a removal of C credits along the entire path.

Once sufficient credit has been removed to meet the original `payment` request, the path stitcher process returns a successful result. However, if the end of the universe list is reached without enough credit being found, the path stitcher process first replaces any credit removed before returning an unsuccessful result.

The path stitching process is fast, since the landmark paths are all pre-computed; the path stitcher process simply walks the paths and removes available credit. Additionally, the path stitcher processes only ever hold a single link lock at a given time, ensuring that Canal is deadlock-free. Finally, the memory requirements of the path stitcher process is low, as it only needs to temporarily store the paths where it has removed credit.

[4]There could be potentially fewer resulting paths, as some of the paths may end up duplicated (e.g., if a path happens to cross two landmarks). This happens rarely in practice.

3.5.5 Implementation

Our implementation of Canal is written in 2,269 lines of C++ (excluding publicly available libraries). The current implementation is designed to be run on a single machine, and uses Pthreads for the universe creator and path stitcher processes and Pthread locks to protect shared data. Our implementation is written so that the deployer can specify the number of universe creator and path stitcher processes to use; the tradeoff depends on the number of incoming `payment` requests that the deployer wishes to process (since more path stitcher processes allows a higher throughput, assuming CPU resources are available). We demonstrate in Section 3.6 that our single-machine implementation is able to support graphs with over 220 million edges.

3.6 Canal microbenchmarks

In this section, we explore a number of Canal microbenchmarks before exploring how Canal performs alongside Sybil tolerance schemes in the following section. Here, we center our evaluation around five questions:

- What is the memory overhead of landmark universes?
- How expensive are landmark universes to compute?
- What is the response time for `payment` requests?
- Do nodes receive all of their available credits over time?
- Do “hotspots" in the network form around landmarks?

Network	Nodes	Links	Avg. degree	Avg. max flow time (s)
Renren [120]	33 K	1.4 M	21.1	0.352
Facebook [193]	63 K	1.6 M	25.7	0.445
YouTube [148]	1.1 M	5.8 M	5.2	2.91
Flickr [147]	1.6 M	30 M	18.8	15.2
Orkut [148]	3.1 M	234 M	76.3	220

Table 3.1: Statistics of the networks we evaluate Canal on. Also included is the average time for completing a max flow computation.

3.6.1 Experimental setup

We evaluate Canal on five real-world social networks of varying size, shown in Table 3.1. These networks are some of the largest publicly available social network data sets, and cover a wide number of nodes (33 K to 3.1 M) and edges (1.4 M to 234 M).

We run our experiments on machines with dual 12-core Intel Xeon X5650 2.66 GHz processors and 48 GB of RAM. In many of the experiments, we vary two key parameters of Canal: the universe level, and the size of the universe list (i.e., the number of universes that are cached). Unless otherwise stated, each experiment is the average of five random trials.

For reference, the final column in Table 3.1 shows the average time to compute max flow[5] between 50 random pairs of nodes in these networks; this further emphasizes the computational expense of conducting credit payments on large networks. Even for networks with only a few million links, the computation time can quickly become multiple seconds.

3.6.2 Memory and compute time of landmark universes

Canal has a certain fixed memory requirement to hold the credit network and edge locks in memory. For example, Orkut, the largest graph we consider, requires almost 20 GB of memory for storing the graph state. However, Canal also requires memory for storing the landmark universes; the amount depends on the universe level and size of the universe list.

[5]We use the push-relabel algorithm [107] for computing max flow.

Figure 3.10 plots the memory size for a single landmark universe of different levels for each of the five networks we consider (multiple landmark universes simply require multiples of this size). We observe that the memory size increases linearly with the landmark universe level, and that the sizes allow multiple landmark universes to be kept in memory on our test machine.

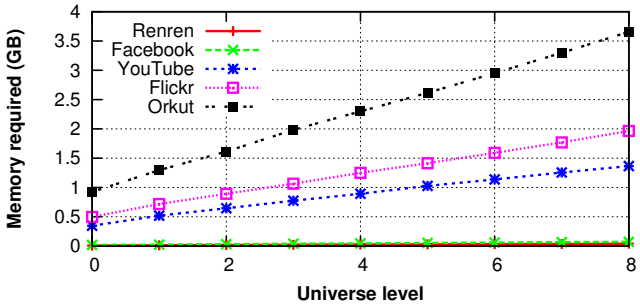


Figure 3.10: Memory requirements of different universe levels. The memory required increases linearly with the universe level.

We now turn to examine how quickly landmark universes can be created. Table 3.2 presents the time required to construct a level-0 landmark universe with a single universe creator process; this follows the general trends of the max flow results in Table 3.1, but is sometimes higher due to Canal's use of per-link locks.

Renren	Facebook	YouTube	Flickr	Orkut
225	292	4,131	13,296	41,787

Table 3.2: Time in milliseconds to calculate a level-0 landmark universe with one universe creator process for various datasets.

However, in Canal, we can take advantage of multiple cores to conduct landmark universe creation in parallel. Figures 3.11 and 3.12 present the speedup (relative to creating a landmark universe with a single universe creator process) and absolute time, respectively, when creating different levels of universes with 22 universe creator processes. We observe that higher level universes enable a greater level of parallelism, meaning Canal is more efficient at creating higher level landmark universes.

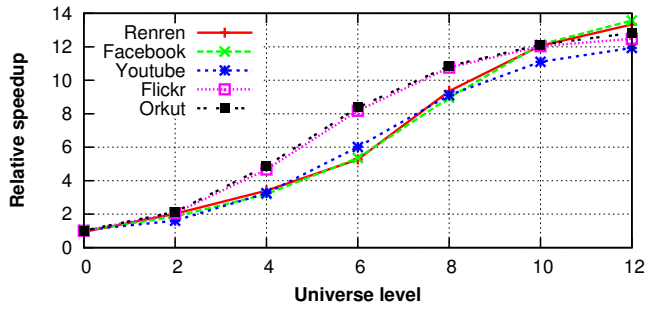


Figure 3.11: Landmark universe creation time speedup, relative to a single universe creator process, for Canal configured with 22 universe creator processes.

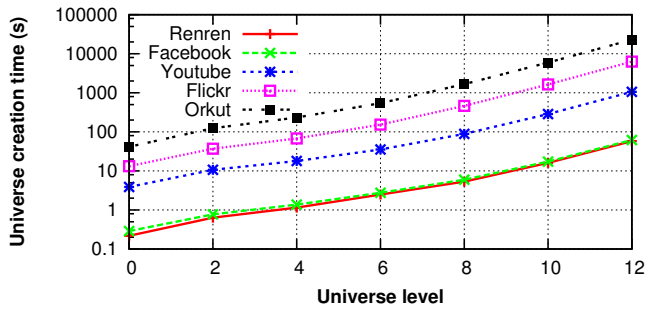


Figure 3.12: Graph showing the absolute landmark universe creation time as we increase the number of universe levels, for Canal configured with 22 universe creator processes.

3.6.3 Latency of `payment` requests

Next, we examine the latency of processing `payment` requests in Canal. For this experiment, we select 5,000 random pairs of nodes in each network, and issue a `payment` request between each pair of nodes for one credit. We then record the latency of the response from Canal. To see how the latency scales with the payment size, we then repeat this experiment by pushing five units of credit. We use 8 level-5 landmark universes, and the credit network is initialized to have one available credit per link.

Table 3.3 presents the results for the five networks. We observe that both the median and 95th percentile latency for pushing one credit is below 1 millisecond for all networks, and the latency for pushing five credits is below 2.5 milliseconds for all networks. This represents a

	1 credit		5 credits	
Network	**Median**	**95th P.**	**Median**	**95th P.**
Renren	0.04	0.09	0.40	0.64
Facebook	0.04	0.13	0.52	0.74
YouTube	0.14	0.59	1.3	2.3
Flickr	0.17	0.51	1.3	2.0
Orkut	0.34	0.83	0.89	1.9

Table 3.3: Median and 95th percentile time in milliseconds taken by Canal to respond to a `payment` requests pushing a one unit and five units of credit.

substantial speedup, as existing systems often take multiple seconds to determine if sufficient available credit is present in the credit network [165, 149].

3.6.4 Do nodes eventually receive all available credit?

Recall that, at any particular moment, Canal can only use a subset of the paths with available credit between two nodes (in particular, it can only use the paths via their common landmarks). However, if available credit along some of these paths is used up, Canal will disregard the exhausted links when constructing new landmark universes. Thus, even though a node only has access to a subset of its available credit at any one time, it will eventually receive more of its credit as new landmark universes are created. We now explore *how long* it takes for a node to access all of its available credit.

To do so, we pick 50 random pairs of nodes with degree greater than 10 (so that there is significant available credit between them) and the credit network is initialized to have one available credit per link. We then conduct a number of *cycles*, where each cycle consists of constructing a new set of 8 level-5 landmark universes and then making the largest payment possible between each pair of nodes (meaning we use up all of the available credit between the two nodes). Over time, we expect that the total payments will approach the actual max flow in the credit network between each pair of nodes.

Figure 3.13 presents the results of this experiment, showing the cumulative fraction of the actual max flow in the credit network that is used for payments. As expected, we see

that the total payments asymptotically approach the actual max flow. More importantly, we observe that it does so very quickly: For example, node pairs in each dataset can achieve between 80% and 95% of their actual max flow in just 4 cycles. This indicates that even though Canal only has access to a subset of paths at any one time, nodes do eventually receive all of their available credit, even over short time windows.

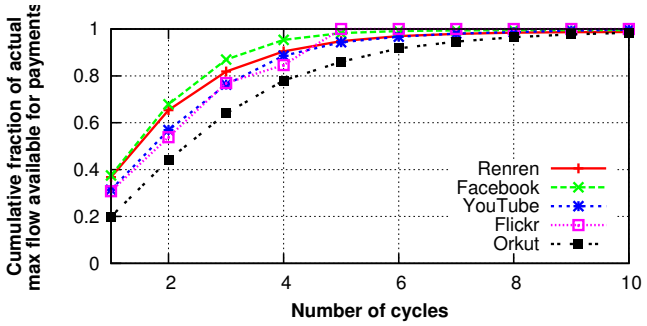


Figure 3.13: Cumulative fraction of actual max flow that is available for payments in Canal, for increasing cycles of landmark universe creation. We observe that nodes can quickly access all of their available credit.

3.6.5 Do landmarks lead to hotspots?

Our final microbenchmark concerns whether or not landmarks in Canal end up becoming "hotspots." To explore this effect, we again select 5,000 random pairs of nodes and have each pair of nodes pay one credit between each other. We then count the total number of times each link in the network was used in transferring a credit. If hotspots form, we would expect to see a number of links used many times. For this experiment, we configure Canal to have 8 level-5 landmark universes.

The results of this experiment are shown in Figure 3.14. We plot the number of times a link is used versus percentile of links, and find that almost all links are used only once, and very few links are used many times. For example, the 90th percentile link is used no more than twice in all networks, and the 99th percentile link is used no more than 14 times.

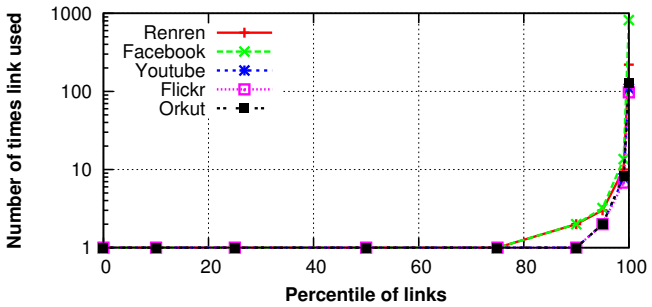


Figure 3.14: Distribution of the number of times links are used when processing 5,000 random credit payments. For all networks, the 99th percentile links are used fewer than 14 times.

3.7 Applying Canal to Sybil tolerance systems

The second half of our evaluation considers the impact that Canal would have on previously proposed Sybil tolerance schemes. In particular, we integrate Canal into both Bazaar [165] and Ostra [149]. We then recreate the original evaluation of these schemes and measure the speedup that these schemes observe when running with Canal, as well as the resulting impact on accuracy (in terms of false negatives).

3.7.1 Bazaar

Recall that Bazaar is designed to strengthen user reputations in online marketplaces like eBay. We replace the storage of the credit network (called the *risk network* in Bazaar) and max flow calculation components with Canal, and re-perform the same evaluation as in the original paper. Bazaar was originally evaluated using a 90-day trace of five of the largest categories on the UK eBay site, and we use the same dataset to evaluate accuracy and speed up of our implementation compared to the original one. The five categories range in size from 419 K users to 1.3 M users, and from 1.2 M links to 5.5 M links as shown in Table 3.4.

Table 3.5 presents the latency for credit network `payment` transactions for the original implementation of Bazaar and for Bazaar augmented with Canal. We make a number of

Category	Nodes	Links
Clothes	1.3 M	5.5 M
Home	1.3 M	4.5 M
Collectables	419 K	1.2 M
Electronics	600 K	1.5 M
Computing	626 K	1.7 M

Table 3.4: Size statistics of the different categories of risk networks used in evaluating Canal implementation of Bazaar.

interesting observations. First, we observe that the median latency for transactions is below 200 microseconds for all categories, and the 95th percentile latency is below 4 milliseconds. This low latency enables Bazaar to be used in an online fashion. Second, when compared to the latency of the original Bazaar implementation, we observe speedups of between 785-fold and 2,329-fold. This underscores the impact of Canal on Sybil tolerance systems like Bazaar.

	Orig.	Canal		Relative
Category	Avg.	Med	95th P.	Speedup
Clothes	6,290	0.2	3.4	2,329 ×
Home	5,340	0.1	3.4	785 ×
Collectables	1,180	0.08	2.0	1,404 ×
Electronics	1,660	0.09	2.70	1,522 ×
Computing	1,410	0.1	2.56	1,084 ×

Table 3.5: Time in milliseconds required to process credit network transactions in Bazaar with Canal with 30 level-2 landmark universes. Also included is the original processing time from the Bazaar paper and the relative speedup. We observe speedups between 785-fold and 2,329-fold.

However, this reduction in latency comes at the cost of accuracy. Since Canal can only look for credit on a subset of paths, it may be unable to find sufficient available credit between a buyer–seller pair, thereby wrongly flagging a transaction as fraudulent. To determine how often this occurs, we calculate the fraction of the transactions for which the original Bazaar implementation found sufficient available credit, but Canal was unable to.

The results of this experiment are presented in Table 3.6, for a configuration with 30 level-3 landmark universes. We observe that Canal provides between 94% and 98%

accuracy in all categories, meaning that at least 94% of the time, Canal is able to find sufficient available credit when the original Bazaar implementation did as well. We further explore the sensitivity of Canal's accuracy to configuration parameters in Figure 3.15, where we vary the number of landmark universes and the level of each universe for the Home category. We observe that accuracy over 95% can be achieved with 20 level-3 landmark universes, suggesting that even a modest number of landmark universes is likely to be sufficient to deploy Canal in practice.

Category	Accuracy
Clothes	94.2%
Home	97.0%
Collectables	97.6%
Electronics	95.4%
Computing	95.9%

Table 3.6: Accuracy of Bazaar implementation using Canal in each category, relative to the original Bazaar implementation. Canal provides high accuracy for Bazaar, implying that users are rarely impacted by the approximate available credit that Canal finds.

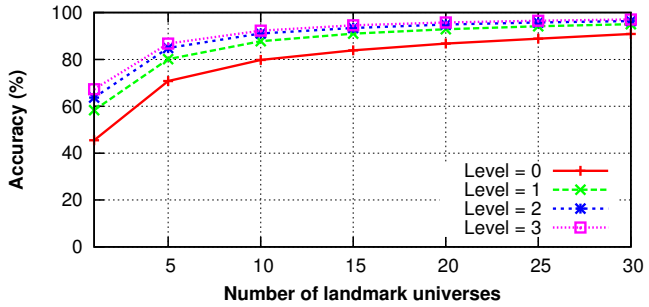


Figure 3.15: Accuracy of Bazaar with Canal for the Home category, for varying numbers of landmark universes and universe levels. Over 95% accuracy can be achieved with 20 level-3 landmark universes.

3.7.2 Ostra

Next, we explore integrating Canal into Ostra [149], a system designed to prevent unwanted communication. Ostra was originally evaluated on a social network derived from largest

strongly connected component of YouTube network [148]; this network consists of 446 K nodes and 3.4 M links [149]. A synthetic communication trace was generated using statistics of a real email trace. We re-run the original Ostra experiments using the same input data, and evaluate accuracy and speed up of our implementation compared to the original one. We use a configuration of Ostra with 128 randomly selected nodes as spammers in the system (each of whom tries to send 500 spam messages), with a credit limit of 3 on every link, and with a 1% false email classification probability by good users.

We first examine the speedup that is observed with Canal deployed to Ostra. Presented in Table 3.7, the results show that if Ostra were to use Canal, the average time taken to find a path with available credit would drop from 35.4 milliseconds to 190 microseconds (a relative speedup of over 186 times). We observe a lower speedup when Canal is applied to Ostra, compared to Bazaar, for two reasons: First, Ostra requires only a single path for every transaction, while Bazaar generally requires the use of multiple paths, and second, the attacker strength is lower for Ostra (just 128 attackers each attempting to send 500 messages).

Original	**Canal**		**Relative**
Avg.	**Median**	**95th Percentile**	**Speedup**
35.4	0.05	1.4	186 ×

Table 3.7: Time in milliseconds required to process a credit network transaction in Ostra in Canal with 30 level-3 landmark universes. Also included is the original processing time from the Ostra paper and the relative speedup.

We now examine the accuracy that Canal provides when deployed with Ostra. Similar to the evaluation with Bazaar, we calculate the fraction of transactions for which the original version of Ostra was able to find a path with available credit, but Canal is not. The results of this experiment for different configurations of the number of landmark universes and the universe level is presented in Figure 3.16. We see that Canal provides over 99% accuracy once at least five landmark universes are used.

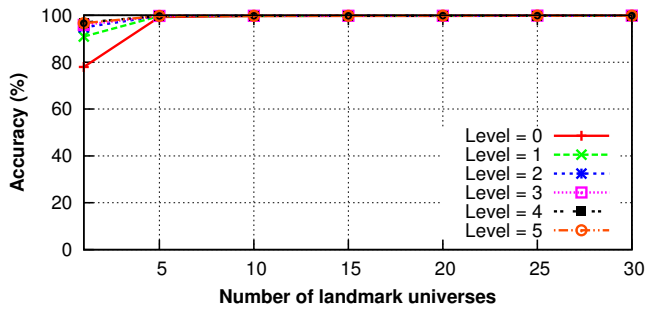


Figure 3.16: Accuracy of the Ostra implementation using Canal, for varying numbers of landmark universes and universe levels. Over 99% accuracy can be achieved once 5 landmark universes are used.

3.7.3 Summary

Overall, the results in this section demonstrate that Canal can be easily integrated into existing Sybil tolerance schemes. Moreover, the results show that implementations of both Ostra and Bazaar with Canal achieve significant speedup while providing an approximation that only rarely impacts the credit available to users. Given the previous high computation cost of these systems, Canal opens the door for schemes like Ostra, Bazaar, and SumUp to be deployed in real systems, with on-demand computations done over highly-dynamic credit networks.

3.8 Discussion: Detection vs. Tolerance

Now that we have presented the details of both social network-based Sybil detection (chapter 2) and Sybil tolerance schemes, we have the opportunity to compare them from the perspective of an operator wishing to deploy these schemes to defend her system from Sybil attacks.

Conceptually, Sybil detection schemes offer a simple model that is easy to integrate with any application. For instance, the system can simply deactivate identities that are classified as likely Sybils and allow all activity from identities classified as non-Sybils. However,

this simplicity and ease of application comes at a high cost for misclassifying an identity as Sybil or non-Sybil. An innocent user who is misclassified (false positive) is denied all service, while a misclassified attacker identity (false negative) is not limited in its malicious activity. Furthermore, existing Sybil detection schemes rely solely on the network structure to identify non-Sybil and Sybil identities, ignoring other relevant information about the activity of identities.

To achieve accuracy, Sybil detection requires the underlying social network to satisfy certain constraints, such as the absence of small cuts within the non-Sybil region (i.e., non-Sybil region should be fast-mixing). Unfortunately, there is mounting evidence that many real-world social networks fail to meet these requirements, either because a significant fraction of their nodes are sparsely connected or their users organize themselves into small tightly-knit communities that are sparsely interconnected. When applied to such networks, Sybil detection schemes suffer from a high rate of misclassified identities.

Credit network-based Sybil tolerance schemes, on the other hand, allow or deny individual transactions among users based on the prevailing system state. This state reflects the history of transactions among users as well as the social graph structure. Thus, Sybil tolerance schemes are deeply embedded in the operation of the system and have to be tailored for each application; they are limited to applications for which an appropriate mechanism is known that lends Sybil tolerance to the relevant system properties.

Sybil tolerance schemes leverage both social network structure and the transaction history, which enables high classification accuracy. Moreover, they allow or deny individual transactions, which leads to a graceful degradation in the presence of false positives or false negatives. It is highly unlikely that all of a legitimate identity's transactions would be blocked due to false positives, or that all of a Sybil identity's transactions would be allowed due to false negatives.

To illustrate these points, consider applying Sybil detection and tolerance schemes to the problem of email spam. Sybil detection schemes would generate a blacklist and whitelist

of Sybil and non-Sybil identities. Any sparsely connected nodes in the fringes of the social network would be blacklisted, while any whitelisted attacker node can send unlimited spam. Sybil tolerance, on the other hand, bounds the rate of spam messages that legitimate users receive from spammers. Sparsely connected legitimate nodes at the fringe of the social network would at worst be limited in the rate at which they can send legitimate messages. Simultaneously, no user has the ability to send an unlimited number of spam messages.

3.9 Limitations and future work

While Sybil tolerance is a significant step towards better social network-based Sybil defense, there are a few opportunities for improvements.

First, as Sybil tolerance schemes rely on both the social network structure and activity information to limit abuse, it is important to understand how a given combination of real world social network topology and a transaction workload would affect credit network liquidity. A full exploration of the necessary connectivity of credit networks and the relationship with the transaction workload remains future work.

Second, while our current implementation of Canal runs on a single machine, we have designed Canal to be implementable across a cluster of machines. Doing so would allow Canal to be deployed on credit networks that are too large to fit in a single machine's memory. Canal can be implemented using graph parallel processing platforms [145, 111] that automatically distribute the graph state (our credit network) across multiple machines. This would mean that the Canal common datastore would be distributed across multiple machines and the universe creator processes would be using distributed graph processing algorithms. Existing graph parallel processing platforms follow the bulk synchronous parallel (BSP) model [190] and the challenge would be to minimize the global communication between the processes and the barrier synchronization cost associated with BSP algorithms. Additionally, a distributed implementation would likely require transactional updates to the common

datastore to properly handle node failures. However, we leave a full implementation of a distributed version of Canal to future work.

3.10 Conclusion

In this chapter, we introduced Sybil tolerance as an approach to social network-based Sybil defense where we limit the impact that a Sybil attack can have on other users, regardless of the number of Sybil identities possessed by the attacker. We showed that Sybil tolerance schemes can be designed using credit-networks and identified the key challenges associated with building credit network-based Sybil tolerance schemes. Next, we focused on the main barrier towards practical deployment of such schemes—high computational complexity associated with performing credit payments over large networks. To this end, we have presented Canal, a system that can efficiently and accurately transfer credit payments over large credit networks. Canal is designed to complement existing Sybil tolerance schemes such as Ostra [149], SumUp [185], TrustDavis [71], and Bazaar [165]. We argued that these schemes are all based on computing payments over credit networks, a computation that requires computing max-flow over a graph, that leads to significant computational complexity and makes them impractical to deploy on real-world sites. With Canal, these schemes see a dramatic speedup, making them practical for real-world use.

CHAPTER 4

Detecting individual misbehaving identities by leveraging activity history

So far, the approaches we discussed require a social network between users. However, not all social computing platforms have an explicit social network or service features that would allow operators to infer a social network (e.g., based on pairwise message exchanges) that satisfies the assumptions made by social network-based Sybil defense schemes (see Chapter 3). This motivates the need for more generally applicable approaches to defend against attacks in social computing platforms. In this chapter, we shift our focus towards a *behavioral profiling* approach, where we leverage past activity history (within the system) associated with identities to detect misbehaving identities. Activity or behavioral history of identities can include all types of interactions that identities have with content in the system (i.e., activity associated with links between users and content such as a user on Facebook liking a page) including interactions along social links (if available).

Recall that the black-market economy for purchasing Facebook likes, Twitter followers, and Yelp and Amazon reviews (widely acknowledged in both industry and academia) is booming today [180, 178, 112, 87, 60]. Customers of these black-market services seek to influence the otherwise "organic" user interactions on the service. It is important to note that they do so through a variety of constantly-evolving strategies including fake (e.g., Sybil) accounts, compromised accounts where malware on an unsuspecting user's computer

clicks likes or posts reviews without the user's knowledge [95], and incentivized collusion networks where users are paid to post content through their account [61, 62].

Behavioral profiling techniques used to address this problem to date have focused primarily on detecting specific attack strategies, for example, detecting Sybil accounts [65, 210, 206], or detecting coordinated posting of content [103]. These methods operate by assuming a particular attacker model (e.g., the attacker is unable to form many social links with normal users) or else they train on known examples of attack traffic, and find other instances of the same attack. Unfortunately, these approaches are not effective against an adaptive attacker. It is known that attackers evolve by changing their strategy, e.g., using compromised accounts with legitimate social links instead of fake accounts [75, 95, 73], to avoid detection.

In this chapter, we investigate a different approach: detecting *anomalous* user behavior that deviates significantly from that of normal users. Our key insight, which we validate empirically, is that normal user behavior in social computing systems can be modeled using only a small number of suitably chosen latent features. Principal Component Analysis (PCA), a technique with well-known applications in uncovering network traffic anomalies [132], can be used to uncover anomalous behavior. Such anomalous behavior may then be subjected to stricter requirements or manual investigations.

We make the following three contributions: First, we introduce the idea of using PCA-based anomaly detection of user behavior in social computing systems. PCA-based anomaly detection requires that user behavior be captured in a small number of dimensions. As discussed in more detail in Section 4.4, using over two years of complete user behavior data from nearly 14K Facebook users, 92K Yelp users, and 100K Twitter users (all sampled uniformly at random), we find that the behavior of normal users in these systems can be captured in the top three to five principal components. Anomalous behavior, then, is user behavior that cannot be adequately captured by these components. Note that unlike prior proposals, *we do not require labeled data in training the detector*. We train our anomaly

detector on a (uniformly) random sampling of Facebook users which contains some (initially unknown) fraction of users with anomalous behavior. Using PCA we are able to distill a detector from this unlabeled data as long as a predominant fraction of users exhibit normal behavior, a property which is known to hold for Facebook.

Second, we evaluate the accuracy of our PCA-based anomaly detection technique on ground-truth data for a diverse set of normal and anomalous user behavior on Facebook. To do so, we acquired traffic from multiple black-market services, identified compromised users, and obtained users who are part of incentivized collusion networks. Our approach detects over 66% of these misbehaving users at less than 3.3% false positive rate. In fact, the detected misbehaving users account for a large fraction, 94% of total misbehavior (number of likes). Section 4.6 reports on the detailed evaluation.

Lastly, in Section 4.7 we apply our technique to detect anomalous ad clicks on the Facebook ad platform. Where only 3% of randomly sampled Facebook users had behavior flagged by us as anomalous (consistent with Facebook's claims [92]), a significantly higher fraction of users liking our Facebook ads had behavior flagged as anomalous. Upon further investigation we find that the like activity behavior of these users is indistinguishable from the behavior of black-market users and compromised users we acquired in the earlier experiment. Our data thus suggests that while the fraction of fake, compromised or otherwise suspicious users on Facebook may be low, they may account for a disproportionately high fraction of ad clicks.

4.1 Related work

First, we discuss prior work related to detecting misbehaving identities. We survey approaches along two axes.

Supervised Learning: A large body of work has explored using supervised machine learning techniques to detect misbehaving identities or malicious content (e.g., spam)

associated with these identities in social computing systems [44, 122, 167, 180, 179, 197, 73, 121, 102, 182, 103]. At a high level, these approaches rely on a labeled dataset of known patterns of normal (honest) behavior and misbehavior to train a machine learning scheme to detect malicious content or identities that exhibit misbehavior. We briefly describe a few of these approaches. Lee et al. [122] propose a scheme that deploys honeypots in social computing systems to attract spam, trains a machine learning (ML) classifier over the captured spam, and then detects new spam using the classifier. Rahman et al. [167] propose a spam and malware detection scheme for Facebook using a Support Vector Machines-based classifier trained using the detected malicious URLs. The COMPA scheme [73] creates statistical behavioral profiles for Twitter users, trains a statistical model with a small manually labeled dataset of both benign and misbehaving users, and then uses it to detect compromised identities in Twitter.

While working with large social computing systems, supervised learning approaches have inherent limitations. Specifically they are attack-specific and vulnerable to *adaptive* attacker strategies. Given the adaptability of the attacker strategies, to maintain efficacy, supervised learning approaches require labeling, training, and classification to be done periodically. In this cat-and-mouse game, they will always lag behind attackers who keep adapting to make a classification imprecise.

Unsupervised Learning: Unsupervised learning-based anomaly detection has been found to be an effective alternative to non-adaptive supervised learning strategies [204, 139, 198, 181, 68] and our work falls in this category. Li et al. [139] propose a system to detect volume anomalies in network traffic using unsupervised PCA-based methods. AutoRE [204] automatically extracts spam URL patterns in email spam based on detecting the bursty and decentralized nature of botnet traffic as anomalous.

In crowdsourcing scenarios, Wang et al. [198] proposed a Sybil detection technique using server-side clickstream models (based on user behavior defined by click-through events generated by users during their browsing sessions in a social computing system). While the

bulk of the paper presents supervised learning schemes to differentiate between Sybil and non-Sybils based on their clickstream behavior, they also propose an unsupervised approach that builds clickstream behavioral clusters that capture normal behavior and users that are not part of normal clusters are flagged as Sybil. However, their approach still requires some constant amount of ground-truth information to figure out clusters that represent normal click-stream behavior. Tan et al. [181] use a user-link graph along with the OSN graph to detect some honest users with supervised ML classifier and then perform an unsupervised analysis to detect OSN spam. CopyCatch [45] detects fraudulent likes by looking for a specific attack signature: groups of users liking the same page at around the same time (*lockstep behavior*). CopyCatch is actively used in Facebook to detect fraudulent likes, however, we show later in Section 4.8 that it is not a silver bullet.

While we welcome the push towards focusing more on unsupervised learning strategies for misbehavior detection, most of the current techniques are quite ad hoc and complex. Our approach using Principal Component Analysis provides a more systematic and general framework for modeling user behavior in social computing systems, and in fact, our PCA-based approach could leverage the user behavior features (e.g., user click-stream models [198]) used in existing work for misbehavior detection.

4.2 Overview

Our goal is to detect anomalous user behavior without *a priori* knowledge of the attacker strategy. Our central premise is that attacker behavior should appear anomalous relative to normal user behavior along some (unknown) latent features. Principal Component Analysis (PCA) is a statistical technique to find these latent features. Section 4.3 describes PCA and our anomaly detection technique in detail. In this section, we first build intuition on why attacker behavior may appear anomalous relative to normal user behavior (regardless of the specific attacker strategy), and overview our approach.

4.2.1 Illustrative example and intuition

Consider a black-market service that has sold a large number of Facebook likes in some time frame to a customer (e.g., the customer's page will receive 10K likes within a week). Since a Facebook user can contribute at most one like to a given page, the black-market service needs to orchestrate likes from a large number of accounts. Given the overhead in acquiring an account—maintaining a fake account or compromising a real account—the service can amortize this overhead by selling to a large number of customers and leveraging each account multiple times, once for each customer. Such behavior may manifest along one of two axes: temporal or spatial (or both). By *temporal* we mean that the timing of the like may be anomalous (e.g., the inter-like delay may be shorter than that of normal users, or the weekday-weekend distribution may differ from normal). By *spatial* anomaly we mean other (non-temporal) characteristics of the like may be anomalous (e.g., the distribution of page categories liked may be different, or combinations of page categories rarely liked together by normal users may be disproportionately more frequent).

A smart attacker would attempt to appear normal along as many features as possible. However, each feature along which he must constrain his behavior reduces the amortization effect, thus limiting the scale at which he can operate. We show in Section 4.6 that black-market users we purchased have nearly an order of magnitude larger number of likes than normal users, and four times larger number of categories liked. If the attacker constrained himself to match normal users, he would require significantly more accounts to maintain the same level of service, adversely affecting profitability.

In the above illustrative example, it is not clear that the number of likes and categories liked are the best features to use (in fact, in Section 4.6.4 we show that such simple approaches are not very effective in practice). Some other feature (or combination of features) that is even more discriminating between normal and anomalous behavior and more constraining for the attacker may be better. Assuming we find such a feature, hard-

coding that feature into the anomaly detection algorithm is undesirable in case "normal" user behavior changes. Thus, our approach must automatically find the most discriminating features to use from unlabeled data.

4.2.2 Approach

At a high level, we build a model for normal user behavior; any users that do not fit the model are flagged as anomalous. We do not make any assumptions about attacker strategy. We use PCA to identify features (dimensions) that best explain the predominant normal user behavior. PCA does so by projecting high-dimensional data into a low-dimensional subspace (called the *normal subspace*) of the top-N principal components that accounts for as much variability in the data as possible. The projection onto the remaining components (called the *residual subspace*) captures anomalies and noise in the data.

To distinguish between anomalies and noise, we compute bounds on the L^2 norm [131] in the residual subspace such that an operator-specified fraction of the *unlabeled* training data (containing predominantly normal user behavior) is within the bound. Note that the normal users do not need to be explicitly identified in the input dataset. When testing for anomalies, any data point whose L^2 norm in the residual subspace exceeds the bound is flagged as anomalous.

4.2.3 Features

We now discuss the input features to PCA that we use to capture user behavior in social computing systems. We focus on modeling Facebook like activity behavior and describe suitable features that capture this behavior.

Temporal features: We define a temporal feature as a *time-series of observed values*. The granularity of the time-series, and the nature of the observed value, depends on the application. We use the number of likes at a per-day granularity. In general, however, the

observed value may be the time-series of number of posts, comments, chat messages, or other user behavior that misbehaving users are suspected of engaging in.

Each time-bucket is a separate dimension. Thus, for a month-long trace, the user's like behavior is described by a ∼30-dimensional vector. The principal components chosen by PCA from this input set can model inter-like delay (i.e., periods with no likes), weekday-weekend patterns, the rate of change of like activity, and other latent features that are linear combinations of the input features, without us having to explicitly identify them.

Spatial features: We define a spatial feature as a *histogram of observed values*. The histogram buckets depend on the application. We use the category of Facebook pages (e.g., sports, politics, education) as buckets, and number of likes in each category as the observed value. In general, one might define histogram buckets for any attribute (e.g., the number of words in comments, the number of friends tagged in photos posted, page-rank of websites shared in posts, etc).

As with temporal features, each spatial histogram bucket is a separate dimension. We use the page categories specified by Facebook[1] to build the spatial feature vector describing the user's like behavior, which PCA then reduces into a low-dimensional representation.

Spatio-temporal features: Spatio-temporal features combine the above two features into a single feature, which captures the *evolution of the spatial distribution of observed values* over time. In essence, it is a time-series of values, where the value in each time bucket summarizes the spatial distribution of observed values at that time. We use *entropy* to summarize the distribution of like categories. Entropy is a measure of information content, computed as $-\sum_i p_i \log_2 p_i$, where bucket i has probability p_i. In general, one might use other metrics depending on the application.

Multiple features: Finally, we note that temporal, spatial, and spatio-temporal features over multiple kinds of user behavior can be combined by simply adding them as extra dimensions. For instance, like activity described using l_T temporal dimensions, l_S spatial dimensions, and

[1]Facebook associates a topic category to each Facebook page which serves as the category of the like.

l_{ST} spatio-temporal dimensions, and wall posting activity described similarly (p_T, p_S, p_{ST}), can be aggregated into a vector with $\sum_x l_x + \sum_x p_x$ dimensions passed as input into PCA.

4.3 Principal Component Analysis (PCA)

Principal component analysis is a tool for finding patterns in high-dimensional data. For a set of m users and n dimensions, we arrange our data in an $m \times n$ matrix $\mathbf{X}$, whose rows correspond to users and whose columns correspond to user behavior features discussed above. PCA then extracts common patterns from the rows of $\mathbf{X}$ in an optimal manner. These common patterns are called *principal components*, and their optimality property is as follows: over the set of all unit vectors having n elements, the first principal component is the one that captures the *maximum variation* contained in the rows of $\mathbf{X}$. More formally, the first principal component v_1 is given by:

$$v_1 = \arg\max_{||v||=1} ||\mathbf{X}v||.$$

The expression $\mathbf{X}v$ yields the inner product (here, equivalent to the correlation) of v with each row of $\mathbf{X}$; so v_1 maximizes the sum of the squared correlations. Loosely, v_1 can be interpreted as the n-dimensional pattern that is most prevalent in the data. In analogous fashion, for each k, the kth principal component captures the maximum amount of correlation beyond what is captured by the previous $k-1$ principal components.

The principal components $v_1, \ldots, v_n$ are constructed to form a *basis* for the rows of $\mathbf{X}$. That is, each row of $\mathbf{X}$ can be expressed as a linear combination of the set of principal components. For any principal component v_k, the amount of variation in the data it captures is given by the corresponding *singular value* σ_k.

A key property often present in matrices that represent measurement data is that only a small subset of principal components suffice to capture most of the variation in the rows of $\mathbf{X}$. If a small subset of singular values are much larger than the rest, we say that the

matrix has *low effective dimension.* Consider the case where r singular values $\sigma_1, \ldots, \sigma_r$ are significantly larger than the rest. Then we know that each row of $\mathbf{X}$ can be approximated as a linear combination of the first r principal components $v_1, \ldots, v_r$; that is, $\mathbf{X}$ has *effective dimension* r.

Low effective dimension frequently occurs in measurement data. It corresponds to the observation that the number of factors that determine or describe measured data is not extremely large. For example, in the case of human-generated data, although data items (users) may be described as points in high-dimensional space (corresponding to the number of time bins or categories), in reality, the set of factors that determine typical human behavior is not nearly so large. A typical example is the user-movie ranking data used in the Netflix prize; while the data matrix of rankings is of size about 550K users $\times$ 18K movies, reasonable results were obtained by treating the matrix as having an effective rank of 20 [128]. In the next section, we demonstrate that this property also holds for user behavior in social computing systems.

4.4 Dimensioning user behavior in social computing systems

To understand dimensionality of user behavior in social computing systems, we analyze a large random sampling of users from three sources: Facebook, Yelp, and Twitter. The Facebook data is new in this study, while the Yelp and Twitter datasets were repurposed for this study from [123] and [46] respectively. We find low-effective dimension in each dataset as discussed below.

4.4.1 User behavior datasets

We use Facebook's people directory [85] to sample Facebook users uniformly at random.[2] The directory summarizes the number of people whose names start with a given character `x`, and allows direct access to the yth user with name starting with `x` at `https://www.facebook.com/directory/people/x-y`. We sample uniformly at random from all possible (1.14B) `x-y` pairs, and follow a series of links to the corresponding user's profile.

We collected the publicly visible like and Timeline [94] activity of 13,991 users over the 26 month period ending in August 2013. For each user, we record three types of features: (i) *temporal*, a time-series of the number of likes at day granularity resulting in 181 dimensions for a 6-month window, (ii) *spatial*, a histogram of the number of likes in the 224 categories defined by Facebook, and (iii) *spatio-temporal*, a time-series of entropy of like categories at day granularity (181 dimensions for 6 months). We compute the entropy H_t on day t as follows: for a user who performs n_t^i likes in category i on day t, and n_t likes in total on day t, we compute $H_t = -\sum_i \frac{n_t^i}{n_t} \log_2 \frac{n_t^i}{n_t}$.

The Yelp dataset consists of all 92,725 Yelp reviewers in the San Francisco area [123] who joined before January 2010 and were active (wrote at least one review) between January 2010 and January 2012. The spatial features are constructed by a histogram of number of reviews posted by the user across 445 random groupings of 22,250 businesses[3] and 8 additional features (related to user reputation provided by Yelp[4]). The dataset also contains temporal features, the time-series of the number of reviews posted by a user at day granularity resulting in 731 dimensions covering the two year period.

[2]Users may opt-out of this directory listing. However, our analysis found 1.14 billion users listed in the directory as of April 2013, while Facebook reported a user count of 1.23 billion in December 2013 [91]. We therefore believe the directory to be substantially complete and representative.

[3]Randomly grouping the feature space helps compress the matrix without affecting the dimensionality of the data [69].

[4]Examples of reputation features include features such as number of review endorsements and number of fans.

The Twitter dataset consists of a random sample of 100K out of the 19M Twitter users who joined before August 2009 [46]. Previous work [46] identified topical experts in Twitter and the topics of interests of users were inferred (e.g., technology, fashion, health, etc) by analyzing the profile of topical experts *followed* by users. In this dataset, each expert's profile is associated with a set of topics of expertise. We construct a spatial histogram by randomly grouping multiple topics (34,334 of them) into 687 topic-groups and counting the number of experts a user is following in a given topic-group. The Twitter dataset does not have temporal features.

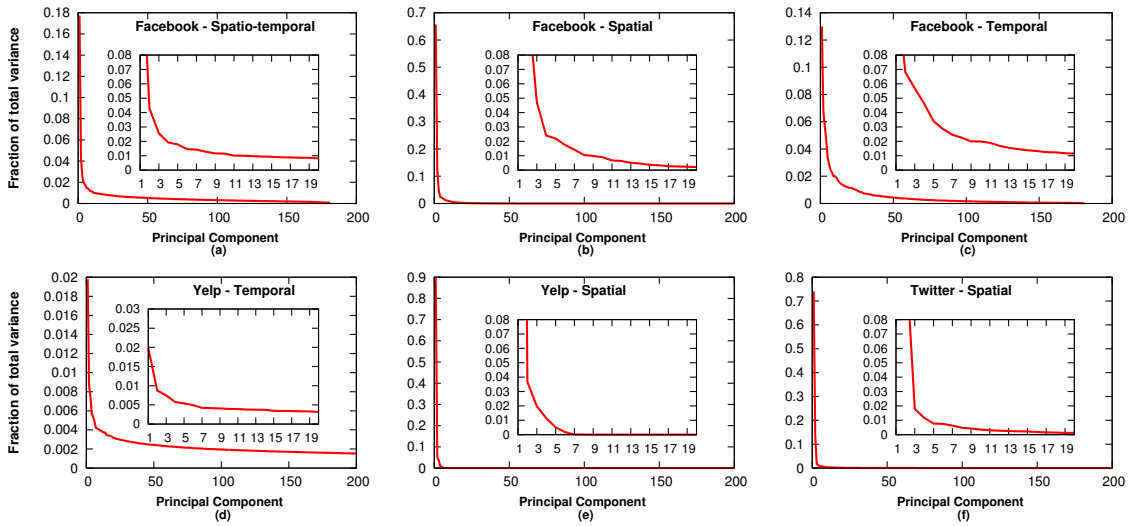


Figure 4.1: Scree plots showing low-dimensionality of normal user behavior. A significant part of variations can be captured using the top three to five principal components (the "knee" of the curves).

4.4.2 Low-dimensionality of user behavior

A key observation in our results from all three social computing systems (Facebook, Yelp, Twitter) across the three user behaviors (temporal, spatial, and spatio-temporal) is that they all have low effective dimension. Figure 4.1 presents *scree plots* that show how much each principal component contributes when used to approximate the user behavior matrix $\mathbf{X}$, and so gives an indication of the effective dimension of $\mathbf{X}$. The effective dimension is the

x-value at the "knee" of the curve (more clearly visible in the inset plot that zooms into the lower dimensions), and the fraction of the area under the curve left of the knee is the total variance of the data accounted for. In other words, the important components are the ones where the slope of the line is very steep, and the components are less important when the slope becomes flat. This method of visually inspecting the scree plot to infer the effective dimension is known as Cattell's Scree test in the statistics literature [54].

For Facebook like behavior (Figure 4.1(a)–(c)), the knee is around five principal components. In fact, for spatial features in Facebook like activity (Figure 4.1(b)), these top five components account for more than 85% of the variance in the data. We perform a parameter sweep in Section 4.6 and find that our anomaly detector is not overly sensitive (detection rate and false positives do not change drastically) to minor variations in the choice of number of principal components [168]. Yelp and Twitter (Figure 4.1(d)–(f)) show a knee between three and five dimensions as well. Overall, across all these datasets where the input dimensionality for user behavior were between 181 and 687, we find that the effective dimensionality is around three to five dimensions.

4.5 Detecting anomalous user behavior

In this section, we elaborate on the normal subspace and residual subspace discussed in Section 4.2, and describe how an operator can use them to detect anomalous behavior.

The operation of separating a user's behavior into principal components can be expressed as a *projection* . Recall that the space spanned by the top k principal components $v_1, \ldots, v_k$ is called the *normal subspace.* The span of the remaining dimensions is referred to as the *residual subspace.* To separate a user's behavior, we project it onto each of these subspaces. Formulating the projection operation computationally is particularly simple since the principal components are unit-norm vectors. We construct the $n \times k$ matrix $\mathbf{P}$ consisting

of the (column) vectors $v_1, \ldots, v_k$. For a particular user's behavior vector x, the normal portion is given by $x_n = \mathbf{P}\mathbf{P}^T x$ and the residual portion is given by $x_r = x - x_n$.

The intuition behind the *residual subspace detection* method for detecting anomalies is that if a user's behavior has a large component that cannot be described in terms of *most* user's behavior, it is anomalous. Specifically, if $\|x_r\|_2$ is unusually large where $\|\cdot\|_2$ represents the L^2 norm, then x is likely anomalous. This requires setting thresholds for $\|x_r\|_2^2$ known as the squared prediction error or SPE [132]. We discuss how we choose a threshold in Section 4.6.

4.5.1 Deployment

In practice, we envision our scheme being deployed by the operator (e.g., Facebook), who has access to all historical user behavior information. The provider first selects a time window in the past (e.g., $T = 6$ months) and a large random sample of users active during that time (e.g., 1M) whose behavior will be used to train the detector. As described earlier, training involves extracting the top k principal components that define the normal and residual subspace for these users. This training is repeated periodically (e.g., every six months) to account for changes in normal user behavior.

The service provider detects anomalous users periodically (e.g., daily or weekly) by constructing the vector of user behavior over the previous T months, projecting it onto the residual subspace from the (latest) training phase, and analyzing the L^2 norm as discussed earlier. Since each user is classified independently, classification can be trivially parallelized.

4.6 Evaluation

We now evaluate the effectiveness of our anomaly detection technique using real-world ground-truth data about normal and anomalous user behavior on Facebook. Our goal with anomaly detection in this section is to detect Facebook like spammers.

4.6.1 Anomalous user ground truth

We collected data for three types of anomalous behaviors: fake (Sybil) accounts that do not have any normal user activity, compromised accounts where the attacker's anomalous activity interleaves with the user's normal activity, and collusion networks where users collectively engage in undesirable behavior. We used the methods described below to collect data for over 6.8K users. We then used Selenium [29] to crawl the publicly visible data for these users, covering 2.16M publicly-visible likes and an additional 1.19M publicly-visible Timeline posts including messages, URLs, and photos. We acquired all activity data for these users from their join date until end of August 2013.

Black-market services: We searched on Google for websites offering paid Facebook likes (query: "buy facebook likes"). We signed up with six services among the top search results and purchased the (standard) package for 1,000 likes; we paid on average $27 to each service. We created a separate Facebook page for each service to like so we could track their performance. Four of the services [78, 79, 80, 81] delivered on their promise (3,437 total users), while the other two [82, 83] did not result in any likes despite successful payment.

As mentioned, we crawled the publicly-visible user behavior of the black-market users who liked our pages. We discovered 1,555,534 likes (with timestamps at day granularity) by these users. We further crawled the users' publicly visible Timeline for public posts yielding an additional 89,452 Timeline posts.

Collusion networks: We discovered collaborative services [61, 62] where users can collaborate (or collude) to boost each other's likes. Users on these services earn virtual credits for liking Facebook pages posted by other users. Users can then "encash" these credits for likes on their own Facebook page. Users can also buy credits (using real money) which they can then encash for likes on their page. We obtained 2,259 likes on three Facebook pages we created, obtaining a set of 2,210 users, at an average cost of around $25 for 1,000 likes. The price for each like (in virtual credits) is set by the user requesting likes;

the higher the price, the more likely it is that other users will accept the offer. We started getting likes within one minute of posting (as compared to more than a day for black-market services).

As with black-market users, we crawled the user activity of the users we found through collusion networks. We collected 359,848 likes and 186,474 Timeline posts.

Compromised accounts: We leveraged the browser malware Febipos.A [95] that infects the user's browser and (silently) performs actions on Facebook and Twitter using the credentials/cookies stored in the browser. The malware consists of a browser plugin, written in (obfuscated) Javascript, for all three major browsers: Chrome, Firefox and Internet Explorer [88, 89].

We installed the malware in a sandbox and de-obfuscated and analyzed the code. The malware periodically contacts a CnC (command-and-control) server for commands, and executes them. We identified 9 commands supported by the version of the malware we analyzed: (1) like a Facebook page, (2) add comments to a Facebook post, (3) share a wall post or photo album, (4) join a Facebook event or Facebook group, (5) post to the user's wall, (6) add comments to photos, (7) send Facebook chat messages, (8) follow a Twitter user, and (9) inject third-party ads into the user's Facebook page.

We reverse-engineered the application-level protocol between the browser component and the CnC server, which uses HTTP as a transport. We then used `curl` to periodically contact the CnC to fetch the commands the CnC would have sent, logging the commands every 5 minutes. In so doing, we believe we were able to monitor the entire activity of the malware for the time we measured it (August 21–30, 2013).

Identifying which other Facebook users are compromised by Febipos.A requires additional data. Unlike in the black-market services and collusion networks— where we were able to create Facebook pages and give to the service to like— we can only passively monitor the malware and cannot inject our page for the other infected users to like (since we do not control the CnC server).

To identify other Facebook users compromised by Febipos.A, we identified two commands issued during the week we monitored the malware: one which instructed the malware to like a specific Facebook page, and second, to join a specific Facebook event. We use Facebook's graph search [86] to find other users that liked the specific page and accepted the specific event directed by the CnC. From this list we sampled a total of 4,596 users. Note, however, that simply because a user matched the two filters does not necessarily mean they are compromised by Febipos.A.

To improve our confidence in compromised users, we clustered the posts (based on content similarity) made to these users' walls and manually inspected the top 20 most common posts. Among these 20 posts, two posts looked suspicious. Upon further investigation, we found out that one of the post was also found on pages the malware was directed to like. The other post was present in the CnC logs we collected. The first was posted by 1,173 users while the second was posted by 135 users. We considered users from both these clusters and obtained a set of 1,193 unique users.[5] We collected 247,589 likes and 916,613 Timeline posts from their profile.

4.6.2 Ethics

We note that all money we paid to acquire anomalous likes were exclusively for pages both controlled by us and setup for the sole purpose of conducting the experiments in this chapter. For the malware analysis, we ensured that our sandbox prevented the malware from executing the CnC's instructions. We did not seek or receive any account credentials of any Facebook user. Overall, we ensured that no other Facebook page or user was harmed or benefited as a result of this research experiment.

[5]The friendship network formed by these users has a very low edge density of 0.00023. Thus, even though they had similar posts on their Timeline, very few of them were friends with each other (further suggesting suspicious behavior).

4.6.3 Normal user ground truth

We collected three datasets to capture normal user behavior. The first dataset is the 719 users that are part of the SIGCOMM [93] and COSN [84] Facebook groups. We picked these technically savvy users, despite the obvious bias, because we presume that these users are less likely to be infected by browser or other malware which we have found to be stealthy enough to avoid detection by non-technically-savvy users. An anomaly detector that has low false-positives on both this dataset as well as a more representative Facebook dataset is more likely to have a range that spans the spectrum of user behavior on Facebook.

For our second dataset, we use the random sampling of Facebook users described in Section 4.4.1. Note that this dataset may be biased in the opposite direction: while it is representative of Facebook users, an unknown fraction of them are fake, compromised, or colluding. Public estimates lower-bound the number of fake users at 3% [92], thus we expect some anomalies in this dataset.

A compromise between the two extremes is our third dataset: a 1-hop crawl of the social-neighborhood of the authors (a total of 1,889 users). This dataset is somewhat more representative of Facebook than the first dataset, and somewhat less likely to be fake, compromised, or colluding than the second dataset. Users in these three datasets in total had 932,704 likes and 2,456,864 Timeline posts putting their level of activity somewhere between the black-market service on the low end, and compromised users on the high end. This fact demonstrates the challenges facing anomaly detectors based on simplistic activity thresholds.

For the rest of the analysis in this chapter, we use the random sampling dataset for training our anomaly detector, and the other two datasets for testing normal users.

Figure 4.2 plots the cumulative distribution (CDF) of likes and comments received on wall posts and the number of *social*[6] posts for all of our six datasets. The top figure plots the

[6]Posts that involve interaction with other users, e.g., photo tagging.

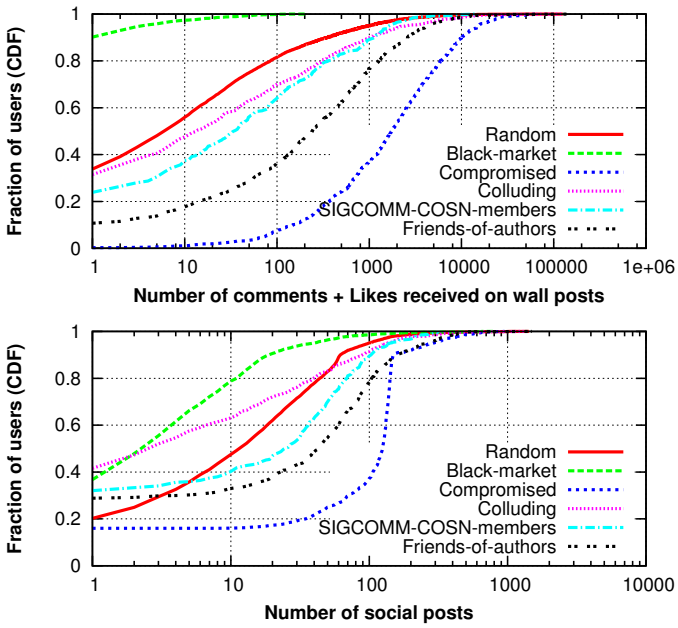


Figure 4.2: Characterizing social activity of normal and anomalous users considered in our study based on activity on their Timeline.

CDF of likes and comments on a logarithmic x-axis ranging from 1 to 1M, and the bottom figure plots the CDF of social posts (messages, URLs, photos). As is evident from the figure, black-market users are the least active, compromised users are the most active, and all three normal user datasets—as well as the collusion network users—fall in the middle and are hard to distinguish visually (especially for social post activity).

4.6.4 Detection Accuracy

Methodology: We analyze Facebook like activity from June 2011 to August 2013. We need to pay special attention to users that joined Facebook in the middle of our analysis period (or stopped being active) to avoid the degenerate case where the anomaly detection flags their lack of activity. We avoid this by considering a six-month sliding window that advances by one month. In each window, we consider users that joined before that window and had at least one like during the window. Unless otherwise mentioned, for the rest of the analysis in this chapter, we consider only these users and their likes that fall within our period of

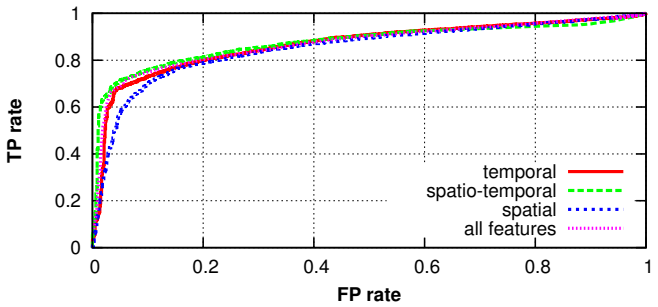


Figure 4.3: ROC curve showing the performance of our anomaly detector in distinguishing between normal and misbehaving users.

analysis—data statistics are shown in Table 4.1. A user's behavior is flagged as anomalous if they are flagged in any one of the sliding time windows. They are flagged as anomalous in a window if the squared prediction error (SPE) exceeds the threshold parameter.

	Random	**Normal**	**Black-market**	**Compro-mised**	**Colluding**
#Users (#likes)	11,851 (561,559)	1,274 (73,388)	3,254 (1,544,107)	1,040 (209,591)	902 (277,600)

Table 4.1: Statistics of different types of users whose like activity (from June 2011 to August 2013) we analyze.

We set the detection threshold (conservatively) based on Facebook's estimate (from their SEC filings [92]) of users that violate terms of service. Facebook estimates around 3.3% users in 2013 to be undesirable (spam or duplicates). Recall that we train our anomaly detector on the like behavior of random Facebook users during much of the same period. We conservatively pick a training threshold that flags 3% of random accounts. We select the top-five components from our PCA output to build the normal subspace.

Results: Figure 4.3 plots the *receiver operating characteristic* (ROC) curve of our detector when evaluated on all datasets for normal and anomalous user behavior (except random, which was used to train the detector) as we perform a parameter-sweep on the detection threshold. The y-axis plots the true-positive rate ($\frac{TP}{TP+FN}$) and the x-axis plots the false-positive rate ($\frac{FP}{FP+TN}$) where TP, TN, FP, FN are true-positive, true-negative,

false-positive, and false-negative, respectively. The area under the ROC curve for an ideal classifier is 1, and that for a random classifier is 0.5. For the mix of misbehaviors represented in our ground-truth dataset, the spatio-temporal feature performs best with an area under the curve of 0.874, followed closely by temporal and spatial features at 0.866 and 0.850, respectively.

By combining the set of users flagged by all three features, our detector is able to flag 66% of all misbehaving users at a false-positive rate of 3.3%. If we compare this with a naïve approach of flagging users based on a simple like volume/day (or like categories/day) cut-off (i.e., by flagging users who exceed a certain number of likes per day or topic categories per day) we can only detect 26% (or 49%) of all misbehaving users at the same false-positive rate. This further suggests that our PCA-based approach is more effective than such naïve approaches at capturing complex normal user behavior patterns to correctly flag misbehaving users.

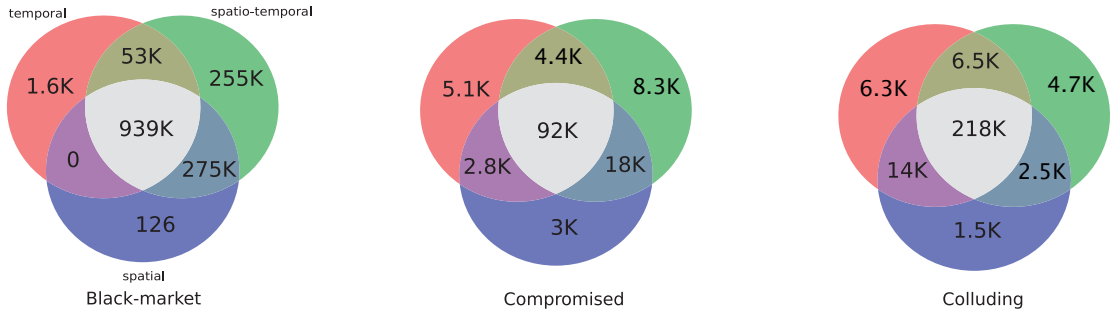


Figure 4.4: Venn diagram illustrating performance of different features in detecting different classes of anomalous user behavior. The numbers indicate number of likes flagged.

Figure 4.4 and Tables 4.2 and 4.3 explore how the set of features performed on the three classes of anomalous behavior. Spatio-temporal features alone flagged 98% of all activity for users acquired through the four black-market services. 61% (939K) of black-market activity was flagged as anomalous by all three sets of features. Due to the dominant nature of the spatio-temporal features on the black-market dataset, there is insufficient data outside the spatio-temporal circle to draw inferences about the other features. The three features

performed more evenly on the dataset of compromised and colluding users, with 43.9% and 78.7% of the anomalous user behavior respectively being flagged by all three sets of features, and 64% and 91% respectively being flagged by at least one. Except in the black-market case, no class of features dominates, and combined they flag 94.3% of all anomalous user behavior in our dataset.

Identity type	**Identities flagged**
Black-market	2,987/3,254 (91%)
Compromised	171/1,040 (16%)
Colluding	269/902 (29%)

Table 4.2: Identities flagged: Performance of different features in detecting different classes of anomalous user behavior.

Identity type	**Likes flagged**			
	Total	**Temporal**	**Spatio-temporal**	**Spatial**
Black-market	1,526,334/1,544,107 (98%)	994,608 (64%)	1,524,576 (98%)	1,215,396 (78%)
Compromised	134,320/209,591 (64%)	104,596 (49%)	123,329 (58%)	116,311 (55%)
Colluding	254,949/277,600 (91%)	246,016 (88%)	232,515 (83%)	237,245 (85%)

Table 4.3: likes flagged: Performance of different features in detecting different classes of anomalous user behavior.

4.6.5 Error Analysis

To better understand our false-negative rate, Figure 4.5 plots the likelihood of detection as a function of the level of activity (number of likes) for each class of anomalous traffic. Unlike black-market users that are easily detected at any level of activity, the anomaly detector does not flag compromised and colluding users with low activity. This is consistent with compromised and colluding user behavior being a blend of normal user behavior intermixed with attacker behavior. At low levels of activity, the detector lacks data to separate anomalous behavior from noise. However, as the attacker leverages the account for more attacks, the probability of detection increases. It increases faster for colluding users, where the user

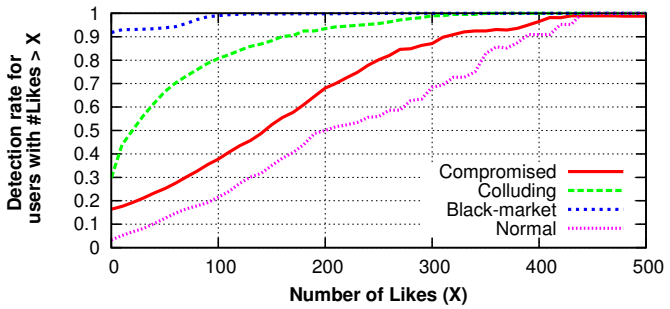


Figure 4.5: Higher like activity generally correlates with higher detection rates, however limits for normal user behavior being flagged are 50–100 likes higher than for anomalous user behavior.

is choosing to engage in anomalous activity, and more slowly for compromised accounts where the user contributes normal behavior to the blend.

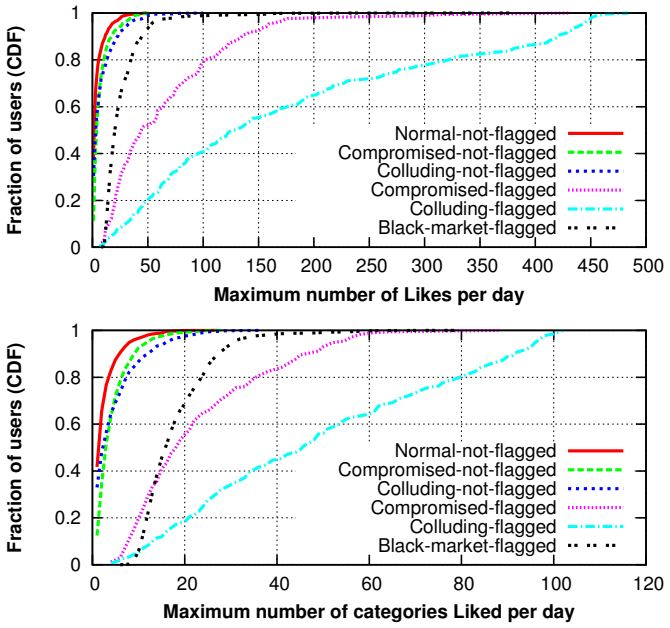


Figure 4.6: Characterizing activity of users that are not flagged in the compromised and colluding set and comparing them with normal users who were not flagged.

Figure 4.6 compares anomalous user behavior that was not flagged by our detector to the behavior of normal users. As is evident from the figure, the false-negatives for compromised and colluding users appear indistinguishable from normal user behavior, especially when compared to the behavior of colluding and compromised users that were flagged. Our

hypothesis (consistent with the previous paragraph) is that these false-negative users are newly compromised users or users newly recruited to the collusion network, and their overall behavior has not yet diverged significantly enough to be considered an anomaly.

Regarding false-positives, we expect some fraction of users to be flagged, since an unknown fraction of the normal users may be infected by malware. We specifically note in Figure 4.5 that the threshold before normal user behavior is flagged is consistently 50–100 likes higher than that for compromised users for the same y-axis value. Thus, our anomaly detection technique accommodates normal users that are naturally prone to clicking on many likes.

4.6.6 Robustness

Next we evaluate the sensitivity of our detector to small variations in the number of principal components chosen for the normal subspace. Figure 4.7 plots the true-positive rate and the false-positive rate as we vary k, the number of principal components used to construct the normal subspace. As is evident from the figure, our detection accuracy does not change appreciably for different choices of k. Thus our detector is quite robust to the number of principal components chosen.

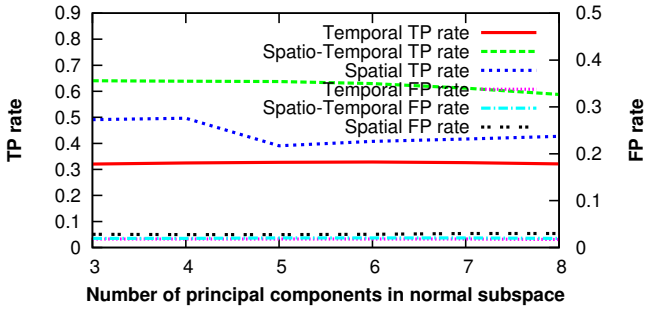


Figure 4.7: False-positive rate and true-positive rate as we vary the number of principal components chosen for the normal subspace. Our detector is stable for small variations in the number of principal components chosen.

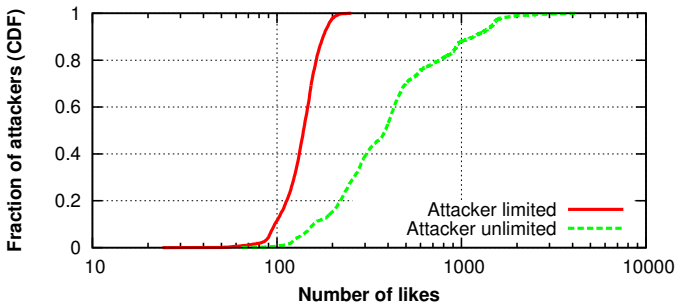


Figure 4.8: Distribution of number of anomalous likes before anomalous users are flagged by our approach. For comparison, we show the actual number of anomalous likes we received.

4.6.7 Adversarial Analysis

In this section, we consider two classes of attackers: first, where the attacker scales back the attack to avoid detection, and second, where the attacker attempts to compromise the training phase.

Scaling Back: Figure 4.8 explores the scenario where attackers scale back their attacks to avoid detection. Specifically, we simulate the scenario where we sub-sample likes uniformly at random from our ground-truth attack traffic (black-market, compromised and colluding) until the point a misbehaving user is no longer flagged by the anomaly detector. As users' behavior spans multiple six month time windows, for each user we consider the window in which the user displayed maximum misbehavior (maximum number of likes in this case). In this way, we analyze the extent to which we can constrain attackers during their peak activity period. We find that our current model parameters constrains attackers by a factor of 3 in the median case, and by an order of magnitude at the 95th percentile.

Compromising Training: An attacker that controls a sufficiently large number of users may attempt to compromise the training phase by injecting additional likes, thereby distorting the principal components learned for normal users [119, 170, 171]. The compromised detector would have a higher false-negative rate, since more anomalous behavior would fall within

the normal subspace. At a high level, this attack may be mitigated by defense-in-depth, where multiple techniques can be used to filter users selected for the training set.

The first defense-in-depth technique is the attacker's need to control a sufficiently large number of anomalous users. We first note that our training data already contains an estimated 3% anomalous users, and that the trained detector has good performance on the ROC curve. Since users in the training set are sampled uniformly at random from all users, an attacker with equivalent power would need to be in control of over $30M$ users (given Facebook's user base of over $1B$ users). In comparison, one of the largest botnets today is estimated to have fewer than 1 million bots [146]. A related issue is that the quantity of like volume that must be injected to affect the detector depends on the overall volume of likes in the system, which is information that is not likely to be readily available to the attacker.

Assuming the attacker is able to amass this large a number of users, the next defense-in-depth technique is to sanitize training data, where anomalous users discovered in one time window are excluded from being used for training in all subsequent time windows [119]. Thus if an attacker ends up altering like traffic significantly in one time window, it could lead to detection and further removal of those anomalous users from the training set.

Finally, variants of PCA that are more robust to outliers can be used to further harden the training phase from compromise. Croux et al. [63, 119] proposed the robust PCA-GRID algorithm that reduces the effect of outliers in the training data. Using this approach one can compute principal components that maximize a more robust measure of data dispersion – the *median absolute deviation* without under-estimating the underlying variance in the data. Such an algorithm could yield robust estimates for the normal subspace.

4.6.8 Scalability

As discussed earlier, classifying users can be trivially parallelized once the training phase is complete. Thus our primary focus in this section is on evaluating the scalability of the training phase.

Space: The total space requirement of the training phase is $O(n \times m)$ where n is the number of input dimensions (typically a few hundred), and m is the number of users in the training set (typically a few million). Thus the space needed to store the matrix is at most a few gigabytes, which can easily fit in a typical server's memory.

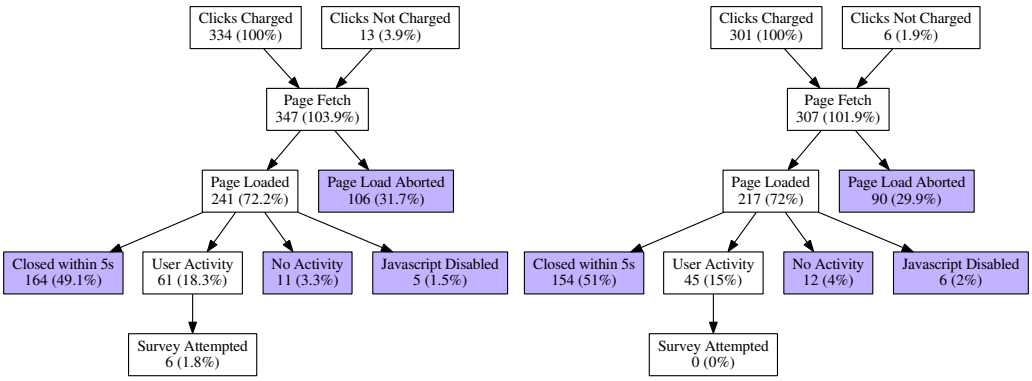
Computation: The primary computation cost in PCA arises from the eigenvalue decomposition of the covariance matrix of the feature vectors, which is a low-order polynomial time algorithm with complexity $O(n^3 + n^2m)$. Eigenvalue decomposition is at the heart of the PageRank algorithm (used in early search engines) for which efficient systems exist to handle input data several orders of magnitude larger than our need [39]. Furthermore, efficient algorithms for PCA based on approximation and matrix sketching have been designed which have close to $O(mn)$ complexity [173, 140].

4.7 Detecting click-spam on Facebook ads

So far, we have discussed the performance of our anomaly detector in detecting diverse attack strategies. Next, we demonstrate another real world application of our technique: detecting click-spam on Facebook ads. Click-spam in online ads—where the advertiser is charged for a click that the user did not intend to make (e.g., accidental clicks, clicks by bots or malware)—is a well-known problem in web search [68, 67], and an emerging problem for Facebook ads [76, 42, 77].

4.7.1 Click-spam in Facebook

To gain a preliminary understanding of Facebook click-spam, we signed up as an advertiser on Facebook. We set up an ad campaign targeting users in the USA aged between 15 and 30. The campaign advertised a simple user survey page about Facebook's privacy settings. When clicked, the ad leads to our heavily instrumented landing page to capture any user activity such as mouse clicks, mouse movement, or keyboard strokes. Of the 334 original



(a) **Original ad** (b) **Bluff ad**

Figure 4.9: Summary of click statistics for real and bluff ad campaigns on Facebook.

ad clicks Facebook charged us for, only 61 (18.3%) performed any activity on the landing page (e.g., mouse move). Figure 4.9(a) shows how users proceeded after clicking the ad. Percentages are relative to the number of ad clicks Facebook charged us for. Shaded boxes are undesirable terminal states that suggest click-spam. For instance, 106 users (31.7%) did not even complete the first HTTP transaction to load the page (e.g., closed the tab, or pressed the back button immediately after clicking the ad).

To distinguish between unintentional clicks and intentional clicks followed by lack of interest in our page, we ran Bluff Ads [116, 67] that are ads with identical targeting parameters as the original ad, but nonsensical content. Our bluff ad content was empty. Figure 4.9(b) shows that our bluff ad performed identically to the original ad, both qualitatively and quantitatively; of 301 clicks in roughly the same time-frame as the original ad, almost 30% did not complete first HTTP, etc. From our data it appears that the content of the ad has no effect on clicks on Facebook ads that we were charged for, a strong indicator of click-spam.

Campaign	Ad target	Cost/like (€)	Total spent (€)	Users		
				Total	Tested	Flagged
1	US	1.62	192.43	119	76	43
2	UK	1.95	230.05	118	69	27
3	AU	0.87	158.89	182	88	38
4	Egypt, Philippines, Malaysia	0.08	47.69	571	261	135
5	India	0.13	30.00	230	199	137
6	India	0.11	22.71	209	169	99
7	India	0.09	22.61	250	199	114
8	India, US, UK	0.22	242.72	1,099	899	791
9	India	0.12	30.00	247	215	143
10	India	0.07	50.00	741	632	372

Table 4.4: Anomalies flagged for different ad campaigns. We observe a significant fraction of anomalous clicks for all campaigns.

4.7.2 Anomalous clicks in Facebook ads

In order to analyze anomalous user behavior, our approach requires information from the user's profile. Due to a change in how Facebook redirects users on ad clicks [129], we were unable to identify the users that clicked on our ad in the experiment above. Fortunately, Facebook offers a different type of ad campaign optimization scheme—maximizing likes—where the destination must be a Facebook page as opposed to an arbitrary website. With such ads, it is possible to identify the users that clicked on such an ad, but not possible to instrument the landing page to get rich telemetry as above. We chose this campaign optimization option for maximizing likes to the advertised page.

We set up 10 ad campaigns, listed in Table 4.4, targeting the 18+ demographic in 7 countries: USA, UK, Australia, Egypt, Philippines, Malaysia and India. Our 10 campaigns were about generic topics such as humor, dogs, trees, and privacy awareness. Our ad contained a like button, a link to the Facebook page, some text, and an image describing the topic of the ad. We ran these ads at different points in time: Campaigns 1 to 4 were run in February 2014, while campaigns 5 to 10 were run in January 2013. In total, we received 3,766 likes for all our pages. For most of the campaigns targeting India (especially #7), we received 80% of the likes within 10 minutes, which is very anomalous.

We first checked whether we obtained most of these likes via social cascades (i.e., a user liking a page because their friend liked it), or from the Facebook ads directly. To do so, we analyzed the edge density of all friendship networks (graph formed by friendship links between users) formed by users of each ad campaign. We find the edge density of friendship networks for all campaigns to be very low (e.g., the friendship network edge density for users in campaign #8 was only 0.000032). This strongly suggests that the Facebook ads, rather than any social cascades, were responsible for the likes.

Out of 3,766 likes, we were able to crawl the identity of the users clicking like for 3,517 likes.[7] Next, we apply our anomaly detection technique from Section 4.5 with the same training data and model parameters that we used in Section 4.6 to 2,767 users (out of 3,517) who fall within our 26-month training window. The penultimate column in Table 4.4 lists the number of users tested in each campaign, and the last column lists the number of users flagged as click-spam.

Of the 2,767 users that clicked our ads in this experiment, 1,867 were flagged as anomalous. Figure 4.10 plots the like activity of the users we flagged as anomalous relative to our normal user behavior dataset, and the black-market user dataset that serves as our ground-truth for anomalous user activity. The flagged users from our ad dataset have an order of magnitude more like activity than the black-market users, and nearly two orders of magnitude more like activity than normal users; they also like twice as many categories as black-market users and almost an order of magnitude more categories than normal users.

4.7.3 Anomaly classification

To better understand the click-spam we observed, we attempt to classify the ad users as one of our three ground-truth anomalous behaviors: black-market, compromised, and collusion. Note that anomaly classification in this section is unrelated to the anomaly detection approach from Section 4.5.

[7] The Facebook user interface does not always show the identity of all users who like a page.

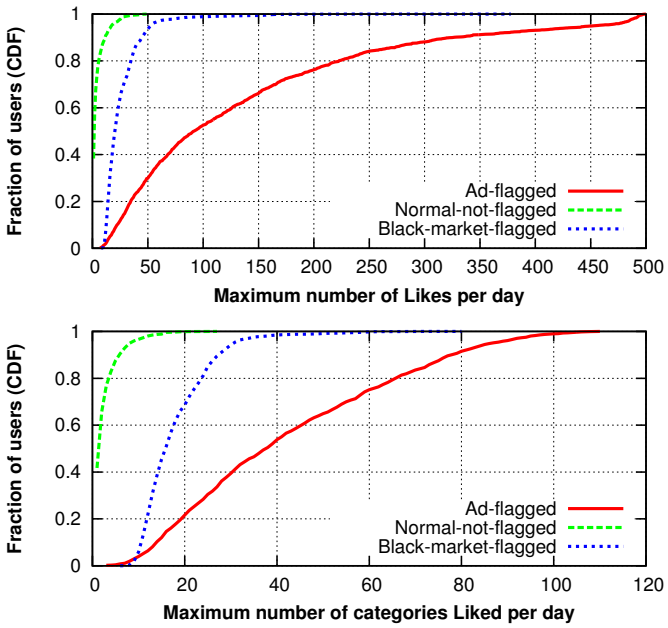


Figure 4.10: Characterizing activity of users flagged in the ad set. Note that most flagged ad users like a much larger number of categories/likes per day than normal and black-market users.

We use the k-Nearest Neighbor (kNN) algorithm for classification. We train the classifier using ground-truth labels for black-market, compromised, and colluding users. The input feature vectors can be formed in different ways: First, we can capture user behavior by projecting it on to the normal and residual subspace. The normal projection reflects normal behavior and the residual projection captures noisy or deviant behavior of a user. Second, we know that user behavior can also be expressed using temporal, spatio-temporal and spatial features. By leveraging all these different combinations, we built 6 classifiers using 6 different feature vectors (2 projections $\times$ 3 features). Each classifier, given an unlabeled user from the ad set, predicts a label for the user.

We use a simple ensemble learning technique of *majority voting* to combine the results of all the classifiers; this also means that there could be test instances that may not be labeled due to lack of consensus. We choose the most recent six-month time window (March to August 2013) in our dataset and use all known misbehaving users (black-market, compromised and colluding) in that window for training the classifier and apply this technique to the 1,408

flagged ad users who fall in that window. To balance classes for training, we randomly under-sample larger classes (black-market and colluding) and use 780 users in each of black-market, colluding and compromised set for training. For each classifier, we pick a value of k that gives the lowest misclassification rate for 10-fold cross validation on the training data. We next apply our trained classifier to predict the unlabeled ad users. Results are averaged over 50 different random trials and we observe an average misclassification rate of 31% (standard deviation of 0.5) based on cross-validation in the training phase. Table 4.5 shows the statistics for the labels predicted for the flagged ad users. We find that the majority of ad users (where we had majority agreement) are classified as black-market or compromised.

Classified As	**Number of users**
Black-market	470
Compromised	109
Colluding	345
Unclassified (no consensus)	484

Table 4.5: Anomaly class predicted for the ad users that are flagged.

While the level of anomalous click traffic is very surprising, it is still unclear what the incentives are for the attacker. One possibility is that black-market accounts and compromised accounts are clicking (liking) ads to generate cover traffic for their misbehavior. Another possibility is that the attacker is trying to drain the budget of some advertiser by clicking on ads of that advertiser. We plan to explore this further as part of future work.

4.8 Corroboration by Facebook

We disclosed our findings to Facebook in March 2014, and included a preprint of our paper [191]. Our primary intent in doing so was to follow responsible disclosure procedures, and to allow Facebook to identify any ethical or technical flaws in our measurement method-

ology. We were informed that Facebook's automated systems detect and remove fake users and fraudulent likes.

Table 4.6 tabulates the users (flagged by our detector) and likes that were removed between the time we conducted our experiments and June 2014. While very few users were removed by Facebook, a sizable fraction of their likes across all pages were indeed removed confirming the accuracy of our detector. To establish a baseline for the fraction of users and likes removed by Facebook's automated systems we find that from our random user dataset (Section 4.4) only 2.2% users, and 32% of all their likes were removed over a ten month period. For black-market, compromised, and colluding users (ground-truth anomalous user dataset from Section 4.6.1), over 50% of all their likes had been removed over 6–10 months. Over 85% of the all likes of users that clicked our ad were removed within four months. Recall that our ad was targeted to normal Facebook users and we did not use any external services to acquire ad likes; nevertheless, 1,730 of the 3,517 likes we were charged for in February 2014 had been removed by Facebook's fraudulent like detection system by June 2014, corroborating our earlier result that a large fraction of users that clicked on our ad are anomalous both by our definition as well as Facebook's.[8] As of this writing we have not received any credit adjustments for the likes charged to our advertiser account that Facebook's fraudulent like detection system since identified and removed.

4.9 Limitations and future work

First, existing defenses that largely focus on detecting individual misbehaving identities have a limitation: when a weak identity has limited or no activity history in the system, defenses lack sufficient information to determine if the identity is misbehaving or honest. In fact, many honest users also tend to have very little or no activity history and it becomes hard to distinguish between misbehaving and honest identities in such cases. Attackers can

[8]While Facebook allows users to un-like pages, according to Facebook insights [90] we had only 56 un-likes across all our pages, which we exclude from our analysis.

	Removed by Facebook's automated systems			
	Users	**likes on all pages**	**likes on our page**	**Timespan**
Normal User Dataset (Section 4.6.3)				
Random users	262/12K	179K/561K	n/a	10 months
Ground-Truth Anomaly Dataset (Section 4.6.1)				
Black-market	228/2987	715K/1.5M	2829/3475	10 months
Compromised	3/171	80K/134K	n/a	7 months
Colluding	9/269	181K/254K	1879/2259	6 months
Facebook Ads Dataset (Section 4.7.2)				
Ad clicks	51/1867	2.9M/3.4M	1730/3517[8]	4 months

Table 4.6: Fraction of users and likes flagged by us removed by Facebook's automated system, as of June 2014.

take advantage of the above limitation to create hard-to-detect Sybil identities with only legitimate or limited past activity. An attacker could stockpile a large number of accounts over a period of time, which can later be used to launch hard-to-detect attacks on the system. We propose a new approach in the next chapter to mitigate abuse without explicitly detecting individual misbehaving identities by analyzing behavior.

Second, most work in building robust defenses stops at the misbehavior-detection phase. From an operator's point of view, for schemes based on detecting anomalous user behavior, the post-detection phase is crucial in discovering new attack patterns (which can be then used to improve the defenses), and also for taking appropriate actions to recover from the abuse. Moreover, the appropriate action might depend on the type of attack. For example, the policy to manage the case of a newly compromised account could be different from that for a fake account that was created solely for abuse purposes. In the future, we plan to investigate data mining techniques to automatically identify different types of attack patterns, given behavioral information about detected suspicious users. This is particularly challenging because an account may not misbehave all the time. One idea would be to separate normal behavior from misbehavior before identifying different attack patterns.

4.10 Conclusion

We propose using Principal Component Analysis (PCA) to detect anomalous user behavior in social computing systems. We use real data from three social computing systems to demonstrate that normal user behavior is low-dimensional along a set of latent features chosen by PCA. We also evaluate our anomaly detection technique using extensive ground-truth data of anomalous behavior exhibited by fake, compromised, and colluding users. Our approach achieves a detection rate of over 66% (covering more than 94% of misbehavior) with less than 3.3% false positives. Notably we need no *a priori* labeling or tuning knobs other than a configured acceptable false positive rate. Finally, we apply our anomaly detection technique to effectively identify anomalous likes on Facebook ads.

CHAPTER 5

Detecting groups of misbehaving identities by leveraging activity history

Popular social computing sites are increasingly employing *crowd computing* to rate and rank content, users, products, and businesses. In such systems, crowd computations involve polling the "wisdom" or "opinions" of crowds—a *group* of users of the system—to provide a variety of recommendation services to their customers. For example, social networking and media sites recommend content based on the number of users who posted or endorsed the content. Similarly e-commerce sites rely on their users to rate and review products and sellers.

Unfortunately, crowd computation systems are vulnerable to Sybil attacks [72], where an attacker creates multiple fake identities with the goal of manipulating the aggregate opinion of the crowd. There are thriving underground markets for launching such tampering attacks on the crowd-sourcing sites mentioned above [155, 180, 191]; typically, the more popular a site, the greater the frequency and magnitude of such attacks. Existing defenses (including our approach discussed in the previous chapter) have mostly taken the approach of detecting individual Sybils, enabling the operator to either suspend the Sybils or nullify their contribution to the crowd computation.

In this chapter, we begin by highlighting a limitation of defenses based on detecting individual Sybil identities: when a weak identity has limited or no activity history (e.g., interactions with other identities or information they post), the defenses lack sufficient

information to determine whether the identity is a Sybil or an inactive non-Sybil. This limitation allows adaptive attackers to create and stockpile large number of Sybil identities with limited prior activity and use them for tampering crowd computations. If the tampered computations involve legitimate content (e.g., promoting a real business on Yelp for a fee, as opposed to promoting malware links on Twitter), it can be hard to detect the tampering (because the act of recommending a real business is by itself not a sign of Sybil activity).

Given the basic limitation of existing defenses, in this chapter, we propose to address Sybil attacks on crowd computations, by moving from *detecting individual Sybil identities* to directly *detecting crowd computations that have significant levels of Sybil identity participation*. Our approach, Stamper, is based on a realization that even when it is fundamentally hard to differentiate between individual Sybil and non-Sybil identities, large *groups* of Sybil and non-Sybil identities can be differentiated. Our approach is based on two key insights:

Key insight 1: If an attacker tampers a computation using a large number of Sybil identities with limited activity, it would result in a *distributional anomaly* or a statistically significant deviation in the distribution of the activity-levels (e.g., number of reviews posted or number of friends formed) of the identities participating in the computation. By analyzing the statistical distributions of the activity-levels of all the identities participating in a crowd computation, we can easily detect such tampering. While there is prior work on detecting malicious activity in crowd computations on e-commerce sites [96, 203] and peer-to-peer search networks [164] that looks for anomalies or specific abnormal patterns in feature distributions (where a feature can be some attribute associated with the user activity), our work stands out by providing improved resilience against adaptive attackers.

Key insight 2: To evade detection by the above insight, a determined attacker would have to forge the activities of the Sybil identities under her control to match the distribution of the activity-levels of non-Sybil identities. To be robust against such adaptive attackers, we employ a novel method: we leverage the key observation that even as the attackers forge the activities of their identities, they *cannot forge the timestamps* of their activities (e.g., join date

or friend link creation time). The timestamp information is typically recorded by operators for all activities of all identities in the system. By analyzing the statistical distributions of the times when the activity-levels of identities have changed, we can significantly raise the bar for evading detection by adaptive attackers (see Section 5.2.3).

Another distinguishing feature of Stamper's tamper detection is that it is agnostic to specific attacker strategies: Unlike existing Sybil detection approaches, Stamper does not make specific assumptions about attacker behaviors; instead it uses anomaly detection to generically detect tampering of any kind. As a result, Stamper can detect computations manipulated by a variety of different attacker strategies, and site operators can choose to further investigate the identities participating in computations flagged by Stamper to detect new and yet undiscovered Sybil identities and attack strategies (see Section 5.4).

We demonstrate the utility and practicality of Stamper approach by evaluating it over data gathered from two widely-used crowd computing systems: Yelp and Twitter. In the case of Yelp, we evaluate accuracy of Stamper in detecting known tampered computations (already identified by Yelp). We demonstrate that Stamper can detect businesses with highly tampered reviews, independently of the strategies attackers used to manipulate the reviews. Using the Twitter dataset, we demonstrate how a site operator can apply Stamper to detect thousands of previously undetected tampered computations in which Sybil identities were used to boost (a) user popularity with Sybil identities following the user and (b) content (tweet) popularity with Sybil identities posting the content. To demo Stamper to our readers, we publicly deployed an online service that applies Stamper over data from the Twitter API to detect tweet content with tampered popularity (`http://trulytweeting.app-ns.mpi-sws.org`).

5.1 Related work and motivation

In Chapter 4, we discussed that there are thriving underground "blackmarket" services, where human users or bots can be "hired" to create Sybil (fake) identities [155, 200]. These Sybil identities are then profitably used to manipulate crowd-sourced information, such as followers in Twitter [7, 183], reviews in Yelp [1], or content likes in Facebook [17].

Limitations with detecting individual Sybil identities: The traditional approach to detect if a computation is manipulated involves determining *which* of the participating identities are Sybils. Identities detected as Sybils are then suspended and their contributions to computations are nullified.

Significant recent research has focused on identifying Sybil identities in the system. A large body of work applied machine learning techniques to distinguish between behaviors (activities and profile characteristics) of Sybil and non-Sybil identities [199, 191, 198, 156, 141, 124, 44] (discussed in Chapter 4). Another body of work has focused on detecting individual Sybil identities by leveraging the structure of the social network graphs formed by Sybils and non-Sybil connecting to one another [194, 49] (discussed in Chapter 2).

However, all these approaches to detect Sybil identities suffer from a limitation: because weak identities are not backed by some external trusted authority, at their core, all Sybil detection schemes have to rely on analyzing an identity's activities (e.g., interactions with other nodes or information they post) to determine if an identity is Sybil. As a result, if an identity has limited or no activity, the schemes lack sufficient evidence to determine if the identity is Sybil or non-Sybil. Studies have shown that many honest users create identities in online systems, but rarely use them [26].

Furthermore, many crowd computations that are tampered with actually involve legitimate content (e.g., promoting a real business on Yelp, as opposed to promoting links to malware sites on Twitter). Since the act of promoting a real business is by itself not mali-

cious, it is even more difficult to detect Sybil identities that tamper only with computations involving legitimate content.

Attackers can take advantage of the above limitations of Sybil detection schemes to create hard-to-detect Sybil identities with only legitimate or limited past activity. An attacker could stockpile a large number of accounts over a period of time, which can later be used to launch hard-to-detect attacks on computations.

Strength in numbers: Given the inherent difficulty in determining whether an *individual identity* is Sybil, we propose to shift the focus to detecting whether a *group of identities* participating in a computation is likely to have Sybil participants.

Few works have explored techniques for preventing Sybil-tampering of computations directly. Prominent among them are DSybil [211], SumUp [185], and Iolaus [123], which work by preventing or discounting votes based on trusted *guides* (DSybil) or the social network (SumUp and Iolaus). Unfortunately, these systems rely on assumptions that do not always hold in a generic crowd-sourcing system. For example, many crowd-sourcing systems do not have social network links interconnecting identities (as assumed by SumUp and Iolaus) and in many systems, a majority of users do not rate many items (preventing the assignment of guides in DSybil).

Prior work has also examined detecting product rating manipulation in e-commerce sites [96, 203] and manipulation of authority scores in peer-to-peer Web search networks [164]. Among them, the most related piece of work is by Feng et al. [96] which explores detecting product rating manipulation in online market places by comparing the distribution of product review scores of an item to known-good distributions. However, unlike Stamper, approach by Feng et al. is vulnerable against adaptive attackers as it only considers distribution of product review scores which can be easily forged to evade detection. Two other works, SynchoTrap [51] and CopyCatch [45] also focused on analyzing behavior of a group of malicious identities. However, they have a similar limitation where they focus on detecting a specific attack behavior: *loosely synchronized actions*, where a group of malicious identities

behave similarly at around the same time. For example, a group of Sybil identities liking a set of Facebook pages at around the same time can be potentially detected by such schemes. In contrast, Stamper does not make any assumptions about specific attacker strategies and thus, has the potential to detect computations tampered using diverse strategies.

5.2 Stamper: Key insights

5.2.1 System model and goal

We consider a crowd computing system (e.g., Twitter, Yelp, Facebook) that uses weak identities for its users. A crowd computation can be voting on a given business by a set of identities in Yelp, or promoting a tweet or following a certain identity by a set of identities in Twitter. A site operator is interested in defending against Sybil attacks on crowd computations within the system.

Goal: For each computation, the goal of the system operator is to determine whether the set of identities participating in the computation included a *sizeable* fraction of Sybil identities. Stamper design focuses on the core challenge of *robustly detecting tampered computations*. The site-specific actions operators might take against the tampered computations are *not* integral to Stamper design. An operator might choose to suspend (remove) the computations detected as tampered or display the computations at the bottom in site-search results or attach warning labels to them.

Reputation scores: We assume that each identity in the system is associated with one or more *reputation scores* that are computed by the operators based on the identity's past activity. Reputation scores can take a variety of forms, and can be computed or obtained by the operators based on "certifications or endorsements", "proofs-of-work", "activity history", or a combination of these. For example, a reputation score could be the number of social network "friends" the identity has in the system, the number of messages it has posted, or the number of endorsements it received from its friends for its work. Given that weak identities

are not backed by external trust, reputation scores reflect the system operators' estimation of trust they would place in the identities in the system, i.e., it is less likely (probable) that identities with higher reputation would misbehave (or be Sybils). Note that by definition all newly created identities (Sybils or non-Sybils) will have zero reputation as they have no prior activity.

Threat model: We assume that an attacker can create arbitrary number of fake identities in the existing system. However, the attacker does not have unbounded economic resources to create and sustain Sybil identities on every newly created site on the Internet. We allow reputation scores to be *forged*, i.e., attackers may manipulate the different reputation scores of malicious identities they control with different amounts of effort. However, we assume that the site operator keeps detailed historical records of the reputation scores of identities over time.[1] The attacker can also obtain the complete historical records of identities' reputation scores; however, the attacker cannot go back in time and tamper with those records.

5.2.2 Detecting tampered computations

We describe how Stamper detects tampered computations in two steps below. We first tackle simple attackers and then consider stronger adaptive attackers.

Step 1: In practice, the distributions of the reputation scores of Sybil and non-Sybil identities tend to be quite different. In other words, some attackers today do not expend significant effort to make their Sybil identities similar to non-Sybil identities. Sybil identities as a group, particularly those with limited or no activity, tend to skew towards low reputation scores (as reputation scores are computed based on the identities' activities on the site), while the non-Sybil identities would naturally span a full spectrum of low to high reputation scores.

[1] Many site operators today including Facebook and Twitter are known to keep detailed historical records of identities.

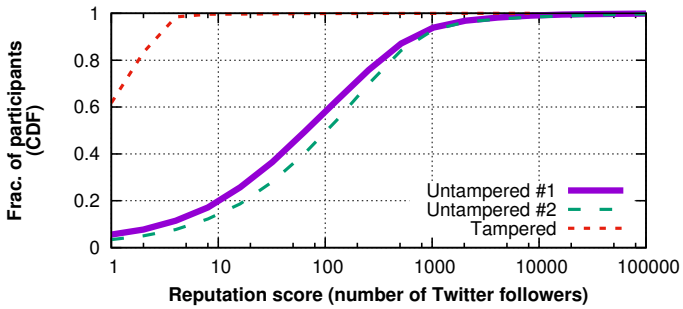


Figure 5.1: Reputation score (based on number of followers) distribution of tampered vs untampered computations. Most participants in the tampered computation have a low reputation score.

Insight 1: Due to the above observation, the participation of Sybil identities in a computation tends to distort the reputation score distributions of the nodes participating in the computation. Figure 5.1 illustrates this insight. It shows the reputation scores for untampered and tampered computations in the Twitter network.[2] The participation of Sybil identities tends to skew the reputation scores towards lower values and has the overall effect of decreasing the entropy in the distribution of scores. It is this difference in reputation score distributions that Stamper leverages to detect Sybil tampering.

We stress that the above insight allows us to determine that a computation has been tampered with even when we cannot determine which of the identities are Sybil. In Figure 5.1, even as we infer that the skew towards lower reputation scores is due to Sybils, we cannot tell which of the identities with low reputation scores are Sybils as there *do* exist non-Sybil identities with such low reputation scores as well.

However, this insight alone would not allow us to design an approach that is robust against an adaptive attacker. For example, a determined attacker could expend additional effort to manipulate the reputation scores of her Sybil identities to match the distribution of reputation scores of non-Sybil identities.

Step 2: Even when an attacker can forge a malicious identity's reputation, she can only forge the *present and future* reputation scores of the identity, but she cannot go back in time

[2]These are samples of real untampered and tampered computations in Twitter flagged by Stamper (See Section 5.4.2).

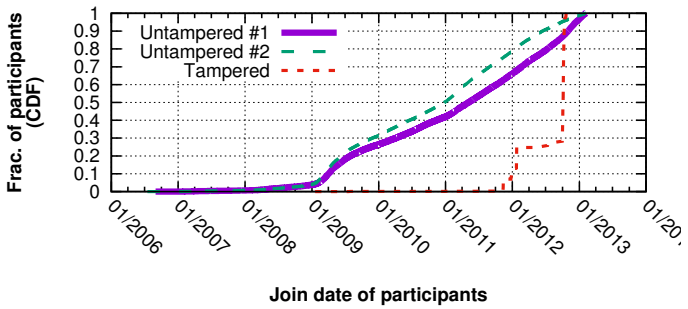


Figure 5.2: Join date distribution of participants of tampered and untampered computation.

and forge the past history (i.e., temporal evolution) of the identity's reputation as recorded by the operator. Thus, the distribution of temporal evolution of forged reputation scores of Sybils tend to exhibit distributions that are quite different than non-Sybils'.

Insight 2: To detect potential forging of reputations of Sybil identities by an adaptive attacker, we analyze the temporal evolution of reputation scores of the identities participating in the computation. Specifically, we examine the distributions of times (i.e., dates) when the identities have achieved a certain percentile (e.g., 0%, 10%, 25%, 50%) of their current reputation score.

Figure 5.2 illustrates this insight. It shows the times when the identities participating in untampered and tampered computations began to acquire reputation in the system (i.e., their join date).[2] The Sybil identities have acquired most of their reputation within the short period of time close to their participation date in the computation, while the non-Sybil identities have acquired their reputations over a much longer period of time. In fact, a significant fraction of untampered computations have identities with reputation histories dating back to the inception of the Twitter site (in 2006). It is this difference in the distributions of temporal evolution of reputation scores of identities that Stamper leverages to detect Sybil tampering of a computation.

5.2.3 Robustness

In this section, we discuss how Stamper raises the bar for evasion by adaptive attackers.

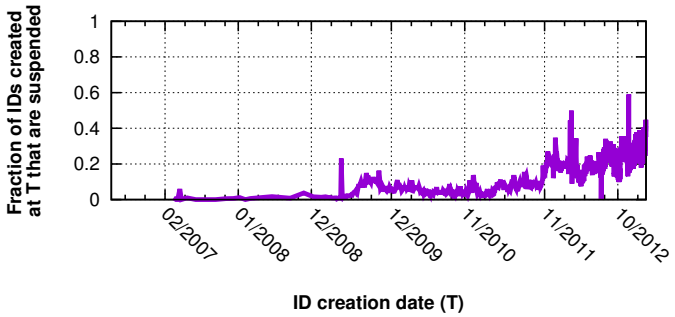


Figure 5.3: Growth in the fraction of identities in Twitter that are eventually suspended.

Robustness 1: Can an adaptive attacker create Sybil identities whose reputation histories match those of non-Sybils? To tamper a computation without being caught by Stamper, an attacker would have to create new Sybil identities and groom their reputations from the inception of the system, i.e., when non-Sybils started being created.

In existing systems like Facebook, Twitter, or Yelp, the attacker is already too late to go back in time and forge identities whose reputation histories date back to the time when these sites came into existence. Figure 5.3 shows the growth in suspended identities in Twitter, since the time of its inception.[3] In the first two years, Twitter witnessed very few (0.036%) malicious identities. However, once Twitter reached a critical mass of users, it started attracting more attackers and the percentage of malicious identities sharply rises to as high as 40% of all identities created on a single day. So, it is hard for any attacker to gain control of Sybil identities with reputation histories dating back to the early years of these existing systems.

An attacker still has an opportunity to create and groom Sybil identities on newly (or yet to be) created online sites. However, the attacker cannot accurately predict which of the several new online sites are likely to succeed and acquire critical mass of users in the long run. So the attacker would have to create Sybil identities on *all* online sites from their inception to be prepared to launch an attack on any single site in the future. Considering the

[3]We crawled 2.3M Twitter identities that joined Twitter at different points of time since its inception. Twitter API allows us to figure out which identities have been suspended.

large number of new online sites that are created everyday, we argue that such attacks are not economically viable in practice.

Robustness 2: Can an adaptive attacker create "sleeper cells" to launch an attack in the distant future? A determined attacker can start creating "sleeper" Sybil identities on popular sites like Facebook and Twitter today, with the goal of launching an attack several years down the road. While such attacks are not impossible, we argue that Stamper significantly raises the costs for the attacker making it economically non-viable. Since Stamper checks for temporal evolution of reputation scores, an attacker would need to actively groom the "sleeper" identities to evolve their reputation scores similar to how non-Sybils reputation scores evolve. As different reputation scores of user identities in different systems evolve in complex, unpredictable ways, an attacker would have to constantly track and mimic these changes. The difficulty of grooming such identities would be reflected in their cost for the attack.

To test our hypothesis about attacker costs for creating identities with high reputation histories, we collected pricing information for Sybil identities in Twitter and Facebook, by manually inspecting postings in 8 online black-market services (that we found via google search), where such identities are sold. A Facebook (Twitter) identity with no reputation costs \$0.51 (\$0.09), while those with 4 years of age cost \$15 (\$1) and those with 5000 (200) real and active friends (followers) costs \$150 (\$5). While the data we gathered does not constitute a rigorous proof, it indicates that identities with long running and high reputations could cost 10 to 100 times or more than newly created Sybil identities with no reputation.

Robustness 3: Can Stamper fundamentally alter the arms race between Sybil attackers and defenses? The root cause of arms race between Sybil attackers and defenses today is that every time a Sybil identity is detected and suspended, attackers can not only create a new Sybil identity and regain their lost attack power, but they can also derive knowledge about how to evade detection. With Stamper, every malicious identity suspended by the site operator would represent a loss in the power of the attacker because the attacker cannot

replace the suspended identities with newly created identities. If used in an attack, Stamper would be able to detect the differences in how reputations of identities evolved over time for older and newer identities.

Most site operators today proactively deploy "spam filters" that over time detect Sybil identities and suspend them. While these spam filters are far from detecting all Sybil identities in a timely manner, Stamper fundamentally shifts the balance of the arms race in favor of these defenses because every suspended identity results in a near permanent reduction in attack power, which can be regained with a new Sybil identity only after waiting for the entire generation of existing identities to leave the system (see Robustness 2 above).

5.3 Stamper Design

We design Stamper to satisfy the following requirements for a practical design: (i) *robustness:* any computation flagged as being tampered with should have been tampered with very high probability and any tampered computation has a good chance of being detected; (ii) *generality:* the system should be able to detect Sybil tampered computations independently of the attack method used.

Notation: Let sets A, M, and H respectively represent all identities, all Sybil identities, and all non-Sybil (honest) identities in the crowd computing system (e.g., Twitter, Yelp, or Facebook) such that $\mathsf{A} = \mathsf{H} \cup \mathsf{M}$. We assume that each identity is associated with a set of reputation scores $\mathsf{R} = \{R_1, R_2, \ldots, R_\ell\}$, which are computed by the operators based on the identity's activity in the system to date. For a given set of identities that participated in a crowd computation c, we denote by $R_i(c)$ to be the probability distribution (or density) function (PDF) of the values of the reputation score R_i of the identities in computation c.

The system operator is interested in defending against Sybil attacks on a set of crowd computations $\mathsf{C} = \{c_1, c_2, \ldots, c_n\}$ Let sets C_i, $\mathsf{M}(C_i)$, $\mathsf{H}(C_i)$ respectively represent all identities, Sybil identities, and non-Sybil identities that are involved in computation c_i.

5.3.1 Design overview

Our goal is to design a "detector" that can check if a given large crowd computation c_i was tampered with by Sybil identities, i.e., whether the (unknown) Sybil identities $\mathsf{M}(C_i)$ constitute a significant fraction of all identities C_i participating in the computation c_i.

We compute the *relative entropy* or divergence between two distributions using a statistical measure called the Kullback–Leibler (KL) divergence [130]. The choice of KL-divergence as the statistical measure is not fundamental to the application of Stamper. We could have used other statistical distance measures [24], but as we show in our evaluation, KL-divergence is quite sufficient for our purposes. KL-divergence ranges from 0 (identical distributions) to ∞ (highly differing distributions); the (symmetric) divergence between two distributions P and Q is denoted $\mathsf{KLD}(\mathsf{P},\mathsf{Q})$, where $\mathsf{KLD}(\mathsf{P},\mathsf{Q}) = \sum_{i=1}^{r}\left(\log(\frac{\mathsf{P}(i)}{\mathsf{Q}(i)})\mathsf{P}(i) + \log(\frac{\mathsf{Q}(i)}{\mathsf{P}(i)})\mathsf{Q}(i)\right)$. Our insight suggests that in practice, the KL-divergence between distributions of untampered computations would be low, while those between untampered and tampered computations would be anomalously high.

5.3.1.1 Detecting anomalous distributions

We use *anomaly detection* techniques [132, 118, 73] to separate out the "outlier" or "anomalous" distributions of reputations scores (and their temporal evolution) observed for tampered computations. Specifically, we apply a variant of anomaly detection known as *semi-supervised* anomaly detection [175], where the site operator has *a priori* knowledge of a small subset of crowd computations, $\mathsf{UC} = \{uc_1, uc_2, \ldots, uc_k\}$ that are largely untampered with by Sybil identities.

We first analyze the KL-divergence in the distribution of a reputation score R_j between the known untampered computations. If we find that the distributions of most untampered computations lie within some small threshold divergence T_j from one another, then we

could declare any other computation whose distribution lies far outside the threshold T_j as potentially tampered.

When identity participation is unbiased: We can offer strong theoretical guarantees on the choice of the threshold T_j, if participants in any untampered crowd computation c_i are drawn uniformly at random (without any bias) from the set of all non-Sybil identities H in the system. Under the unbiased participation assumption, the probability distributions of the reputation scores of identities participating in all large untampered computations are guaranteed to converge to the same distribution. Specifically, as the size of a computation c_i grows, the distribution of reputation scores $R_j(c_i)$ quickly approximates the distribution of reputation scores for all non-Sybil identities $R_j(\mathsf{H})$. Formally, for all c_i such that $C_i = \mathsf{H}(C_i)$, and for some small ϵ,

$$\exists\, s \;\; \text{s.t.} \;\; \forall i, |C_i| > s \;\cap\; \mathsf{KLD}(R_j(\mathsf{H}), R_j(C_i)) < \epsilon.$$

We refer to s as the *size threshold* for the crowd computations. In fact, Roy [169] studied the thresholds theoretically as well as empirically and proved an upper bound of $1/|C_i|$ on KLD for a sampled distribution of size $|C_i|$. Thus, if an untampered computation involves over 100 or 1,000 identities, the KLD between the reputation score distributions of the computation participants and the non-Sybil identities will be lower than 0.01 or 0.001, respectively. As a result, a simple strategy for detecting whether a given large computation c_i (i.e., $|C_i| > s$) has been tampered with is as follows: First, select some *a priori* known untampered computation c_u of size greater than s. Then, compute the divergence in the distributions of reputation score R_j between the given computation and known untampered computation, i.e., compute $\mathsf{KLD}(R_j(c_u), R_j(c_i))$. If the divergence is greater than the divergence threshold $1/s$, declare the computation c_i as tampered (with high probability).

When identity participation is biased: In practice, many crowd computations draw a biased population of identities: For example, in Yelp, many reviewers of businesses in San

Francisco are likely to be drawn from San Francisco. Without the unbiased participation assumption, we cannot offer any *theoretical* guarantees on convergence of distributions of untampered computations. However, in practice we often observe that the *distributions for untampered computations are far closer to one another than they are to tampered computations*. We validate this claim using real-world data from Yelp and Twitter in the evaluation Sections 5.4.1 and 5.4.2.

In the case of biased participation, we first derive a *reference* or expected distribution by "averaging" the distributions of known untampered computations and then select a KL-divergence threshold T_j that encompasses most, if not all, the untampered computations. To detect whether a given large computation c_i is tampered with, we compute its KL-divergence from the reference distribution. If it is larger than the threshold divergence T_j, we declare the computation c_i as tampered (with high probability).

While we defer the precise details of the threshold selection to Section 5.3.2, we make two observations on the choice of the divergence threshold. First, if for some reputation score R_j, the distributions of untampered computations do not converge in practice, then the observed threshold divergence T_j between the untampered computations would also naturally be quite large, and consequently there would be little risk of an untampered computation flagged as tampered. Thus, the risk of untampered computations being flagged as tampered is low, even when the distributions of untampered computations do not converge. Second, by raising and lowering the threshold T_j, an operator can trade-off between the precision and recall in detecting tampered computations. Depending on the application scenario, operators can either choose a more or less conservative threshold.

5.3.2 Detailed Design

The operator would deploy Stamper as follows:

1. Creating a pool of reputation scores: The first step in deploying Stamper involves choosing a set of reputation scores $\{R_1, R_2, ..., R_l\}$ that can be computed for each identity in

the system. Identities start with low (zero) reputation scores when they are created and can earn higher reputations over time. We do *not* assume that reputation scores are unforgeable: different reputation scores of an identity may be manipulated by the attacker with different amounts of effort.

2. Building a reference (expected) distribution: To build a reference distribution for a given reputation score R_j, we first compute the distribution of the reputation score for each known-untampered computation (i.e., calculates $R_j(uc_i)$ for each $uc_i \in \mathsf{UC}$). Now, these distributions are aggregated into a single reference distribution $\mathsf{R}_j(\mathsf{UC})$ using a *linear opinion pool* [59] model. We do so using a fair weighting scheme such that each crowd computation contributes a fair share towards building the final reference distribution. Formally, the reference distribution $\mathsf{R}_j(\mathsf{UC})$ is defined as $Pr[v \leftarrow \mathsf{R}_j(\mathsf{UC})] = \frac{1}{k}\sum_{i=1}^{k} Pr[v \leftarrow \mathsf{R}_j(uc_i)]$

3. Selecting a threshold: We now compute the KL-divergence of each of the crowd computations in set C from that of the reference distribution. We will obtain a range of KL-divergence values and will select a threshold T_j, such that KL-divergence values greater than T_j is anomalous with respect to the rest of KL-divergence values. To select this threshold, we use a simple statistical technique called the *box plot rule* [161] defined as follows: Let $Q1$ and $Q3$ be the lower and upper quartile respectively, for the KL-divergence values. A KL-divergence value is an outlier if it lies beyond the *upper outer fence:* $Q3 + 3 * (Q3 - Q1)$. We select the upper outer fence of the distribution as the threshold T_j.

4. Detecting anomalous computations: With T_j and $\mathsf{R}_j(\mathsf{UC})$, it is straightforward to detect anomalous computations. For a given computation c_i, the operator simply calculates the KL-divergence between $R_j(c_i)$ and $\mathsf{R}_j(\mathsf{UC})$; if it is higher than T_j, the computation is flagged as anomalous. In fact, the higher divergence (above the threshold), the more anomalous the computation turns out when compared to the rest of the computations. The operator can experiment with the tradeoff of catching more tampered computations (when using a lower KL-divergence threshold) versus improving the efficiency of workers (when using a high KL-divergence threshold).

Operators typically use human workers to examine suspicious accounts or activities once they are flagged by their defense mechanisms [50]. Stamper can guide operators to focus the attention of their human verifiers on a set of flagged computations to verify if they are tampered with. More importantly, while Stamper has been designed to detect computation tampering, it can be used in practice for a broader range of Sybil defense tasks. The operator can manually investigate the anomalous computations—as they have a higher chance of containing Sybils—to further discover new Sybils and previously unknown attacker strategies. We demonstrate this in Section 5.4.2.2 where we investigate the identities that participate in tampered computations. However, it should be noted that an investigation phase is common in deployed defense schemes and it is not part of the core Stamper deployment workflow.

5.4 Stamper Evaluation

This section presents three case studies of applying Stamper in two widely used crowd-sourcing systems, namely Yelp and Twitter. The goals of Stamper evaluation is three fold: (1) How easy or difficult is it to deploy Stamper to detect tampered computations in crowd-sourcing systems? All three case studies using data from Yelp and Twitter try to address this question. (2) Can Stamper detect known tampered computations with high accuracy? We leverage the Yelp dataset to evaluate accuracy of Stamper on tampered computations previously detected by Yelp. (3) Can Stamper detect tampered computations that are still undetected on crowd-sourcing sites? We present two case studies in Twitter, where we apply Stamper to detect previously unknown tampered computations and also uncover new (previously unknown) Sybil attacker strategies and new (previously unknown) Sybil identities in the system.

5.4.1 Case 1: Yelp review tampering

Goal: Find businesses with tampered reviews: Yelp is a popular local directory service, where users can search for businesses in a given locality and retrieve crowd-sourced reviews and ratings for those businesses from other users. As Yelp is becoming popular, businesses (e.g., restaurants) have an incentive to manipulate their reviews and ratings in their favor. Today, there are plenty of black-market services [5], where one can easily buy Yelp reviews for a cheap price. The crowd computation that we are interested in is the set of identities that rate a given business in Yelp. Our goal is to evaluate how effectively Stamper can be leveraged to detect attacks that tamper the computation, i.e., detect businesses that have manipulated reviews.

For evaluating effectiveness of Stamper we leverage Yelp's *review filter* [33, 30] feature to obtain "ground truth" for tampered reviews. Yelp filters suspicious reviews to defend against fake reviews. It should be noted that as is the case with many online defense schemes deployed today, Yelp acknowledges that their review system is not perfect and may not be able to detect all types of tampered reviews and may even sometimes wrongly flag legitimate reviews. However, for the purpose of this analysis, we will consider a business to have tampered reviews if Yelp filters at least one review of the business. Note that we do not have any knowledge about specific strategies used by attackers of the filtered reviews (i.e., did the attacker create multiple fake accounts to tamper reviews or did she incentivize real users to write fake reviews in return for a monetary reward).

More precisely, we investigate the following three questions: (i) How easy or difficult is it to apply Stamper to detect computation (review) tampering in Yelp? (ii) Does the key requirement that distributions of reputation scores of large untampered computations converge (while those of large tampered computations diverge) hold in practice in Yelp? (iii) Can Stamper detect most of the *highly tampered* computations (businesses with a majority of reviews filtered) at a low false positive rate (fraction of businesses with no filtered

reviews flagged)? Stamper is designed to detect *highly tampered* computations and cannot guarantee detection of computations tampered only to a small extent (see Section 5.5).

Data gathered. We used Yelp data gathered by Kakhki et al. [123] in May 2012, which we updated with our own data gathering crawl in March 2013. This dataset consists of all businesses on Yelp in the city of San Francisco at that time. This includes 30,339 businesses with a total of 1,655,385 ratings from 340,671 reviewer identities. Each rating consists of a score from 1 to 5 stars. These ratings also include those *filtered* by Yelp's automated review filter. In total, Yelp filtered 195,825 (or 11.83%) ratings. As Stamper has been designed to infer tampering in large computations involving more than a certain number of identities, we threshold the size of the computation at 100 for Yelp. There are 3,579 businesses with more than 100 reviews. Out of these 3,579 businesses, there are 54 businesses which did not have a single review filtered by Yelp. We consider these 54 cases as untampered computations. Also, for each reviewer we collected information about a variety of reputation scores. (See the first column of Table 5.1.)

5.4.1.1 Ease of deploying Stamper

The four steps that constitute Stamper detection strategy (outlined in Section 5.3.2) can be applied in a straight-forward manner with very little overhead.

1. Creating a pool of reputation scores: The first column in Table 5.1 lists all the 8 reputation scores used in our evaluation; e.g., the reputation score in the 8th row is a measure of the number of times reviews by an identity are marked useful by other identities in the service. To tamper a crowd computation by forging this reputation score, an attacker would have to put additional effort to boost the reputation for the malicious identities employed in the attack by obtaining a certain number of endorsements (by getting reviews marked useful) from other identities.

2. Building reference distributions: We select businesses which had no (zero) review filtered as the set of untampered computations. There are 54 such businesses (with zero reviews

Reputation score	# flagged	Percentage of computations flagged				
		0% filtered	(0,10]% filtered	(10,30]% filtered	(30,50]% filtered	> 50% filtered
# photos	158	**5.6 (3/54)**	0.2 (5/2280)	6.6 (72/1089)	32.9 (27/82)	**68.9 (51/74)**
# first badges	141	**0.0 (0/54)**	1.3 (30/2280)	4.4 (48/1089)	20.7 (17/82)	**62.2 (46/74)**
# fans	139	**0.0 (0/54)**	0.6 (14/2280)	5.0 (54/1089)	28.0 (23/82)	**64.9 (48/74)**
# compliments	173	**1.9 (1/54)**	0.7 (15/2280)	6.2 (67/1089)	35.4 (29/82)	**82.4 (61/74)**
# reviews marked funny	157	**0.0 (0/54)**	0.7 (15/2280)	5.0 (54/1089)	28.0 (23/82)	**87.8 (65/74)**
# reviews marked cool	174	**0.0 (0/54)**	1.0 (22/2280)	5.6 (61/1089)	35.4 (29/82)	**83.8 (62/74)**
# friends	227	**0.0 (0/54)**	0.2 (4/2280)	9.7 (106/1089)	56.1 (46/82)	**95.9 (71/74)**
# reviews marked useful	224	**3.7 (2/54)**	0.5 (11/2280)	9.2 (100/1089)	51.2 (42/82)	**93.2 (69/74)**
All scores combined	362	**5.6 (3/54)**	3.0 (68/2280)	14.8 (161/1089)	70.7 (58/82)	**97.3 (72/74)**

Table 5.1: Computations with varied levels of filtered reviews flagged by Stamper. Stamper flags most of the *highly tampered* (>50% filtered) computations while flagging very few (3/54) untampered computations.

filtered). Even though the number of untampered computations might seem small, they have a large number of reviewers (14,223 reviewers) who wrote reviews for them.

3. Selecting a threshold: We compute KL-divergence values for all 3,579 businesses from the reference distribution for each reputation score. Then, for each reputation score, using the box plot rule, we estimate a KL-divergence threshold for flagging anomalies; e.g., in the case of the reputation score, *#times review is marked useful* (we will call this as the number of review endorsements), we estimate a threshold of 1.2.

4. Detecting anomalous computations: If the KL-divergence computed for a business is greater than the divergence threshold for any reputation score, the computation is marked as anomalous. The second column of Table 5.1 shows the number of businesses whose reviews have been flagged as tampered by Stamper.

5.4.1.2 Detectability of untampered computations

We now investigate whether a key assumption behind Stamper design holds in practice. Specifically, we verify if the distributions of reputation scores of large untampered computa-

tions in Yelp tend to converge, while those of tampered computations diverge. Figure 5.4 shows the distribution of KL-divergence values using the endorsement count reputation score for untampered computations and computations with different levels of filtered reviews. Note that, computations with zero and with less than 10% reviews filtered show low KL-divergence values from the reference distribution, indicating a good convergence in the reputation score distributions. In fact, for 90% of untampered computations their divergence values are less than or equal to 0.36. While for tampered computations (computations with more than 20% and 50% reviews filtered), the KL-divergence values are higher and shows a diverging trend. We observe a similar trend for other reputation scores listed in Table 5.1.

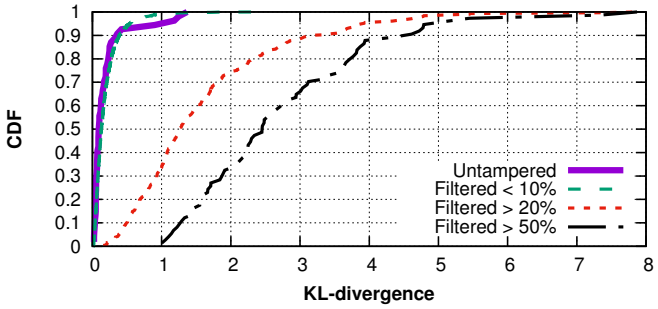


Figure 5.4: Distribution of KL-divergence values for untampered and tampered computations using number of review endorsements in Yelp.

5.4.1.3 Robustness of Stamper tamper detection

Next we investigate the robustness of Stamper detection. For the rest of the analysis, we divide businesses into five categories based on the level of filtering: 0% filtered (or untampered), 0 to 10%, 10 to 30%, 30 to 50%, and more than 50% filtered. We consider computations with more than 50% reviews filtered to be highly tampered. Out of a total of 3,525 businesses with at least one filtered review, there are 74 businesses that are highly tampered.

1. Stamper can detect most of the highly tampered computations: The last column in Table 5.1 shows the fraction of highly tampered computations that are flagged. By combining all 8 reputation scores (a computation is flagged if it is flagged by at least one reputation

score), we detect more than 97% of highly tampered computations. It is interesting to note that combining multiple reputation scores can help to catch more tampered computations. Stamper also manages to catch a significantly high fraction (over 70%) of computations with 30 to 50% reviews filtered.

2. Stamper has low false positives: The third column in Table 5.1 shows the fraction of untampered computations flagged. By combining all 8 reputation scores, we observe a false positive rate of only 5.6%. While interpreting this false positive rate, it is important to keep in mind that Yelp's review filter is not perfect and could have potentially missed flagging some fake reviews.

5.4.1.4 Discussion

Why is Stamper useful for a system like Yelp? Note that Stamper does not detect individual suspicious reviews. While this might sound like a limitation, Stamper can still be useful for Yelp in flagging businesses with highly tampered reviews. For example, Yelp is known to suspend businesses that were caught buying reviews [2], and display a warning when a user visits a business page suspected of tampering reviews [32]. Using Stamper, Yelp can do so even without detecting individual suspicious reviews as they might be very hard to detect for various reasons. For example, Yelp went to the extent of conducting sting operations to catch businesses trying to buy fake reviews [8] because we suspect that such type of tampering is very hard to detect by analyzing reviewer behavior or the content of their reviews. With Stamper, Yelp has the potential to catch highly tampered computations even with very minimum or no information about the behavior of the reviewers. Another huge advantage of our scheme is that compared to prior machine learning approaches, Stamper can detect highly tampered computations in Yelp without training on any pre-identified fake reviews.

Leveraging temporal evolution of reputation scores: We tried to find anomalous computations by analyzing the temporal evolution of reputation scores. We used the timestamp at

0th percentile reputation, which is the join date of the user. Stamper flagged only 2 highly tampered computations (already caught by the other reputation scores) using join dates. We suspect that attackers on Yelp are not trying hard to forge their reputation scores today, and we are able to detect most of the highly tampered computations using simple reputation scores. In the next two case studies, we observe that temporal evolution of reputation scores are very helpful in catching adaptive attackers.

5.4.2 Case 2: Twitter follower tampering

Goal: Find Twitter users with fake followers: In Twitter, to obtain real time information posted by specific users, users typically *follow* those users. Today, the influence of a user is often estimated by counting the number of followers. As a result, there are strong incentives for users to acquire more users to follow them and there have been numerous reports of follower count manipulations [178]. Thus, the crowd computation that we are interested in is the set of identities in Twitter that follow a given Twitter identity. Our goal is detect attacks that manipulate the computation, i.e., detect identities that have tampered follower counts.

We use this case study to showcase Stamper's capability of detecting yet unknown tampered computations. This provides an opportunity to evaluate how system operators (who in practice would not have a priori ground truth information about tampered computations) might use Stamper. More precisely, we investigate the following three questions: (i) How easy or difficult is it to apply Stamper to detect computation (follower count) tampering in Twitter? (ii) Can system operators analyze the computations flagged by Stamper further (potentially manually) to detect (potentially new) patterns of Sybil attacks? (iii) Can the newly discovered Sybil attack patterns be used to uncover more Sybil identities?

Data gathered: We target detecting tampering of follower-counts only for popular Twitter user identities with more than 1,000 followers. We obtained the Twitter-UIDs (unique identifiers) of all users with more than 1,000 followers in all of Twitter (as of July 2012)

from a research group which collected this data for a separate study [101]. This dataset contained 2,100,851 identities. We selected a random sample of 70,000 of these identities and gathered profile information of all their followers. Some of these accounts no longer existed on Twitter and their information could not be collected. In total, we discovered (in aggregate) over 176M followers for 69,409 of these users.

5.4.2.1 Ease of deploying Stamper

We briefly discuss the steps that constitute Stamper procedure for this case study.

1. Creating a pool of reputation scores: We select *number of followers* as a reputation score to build our reference distribution (i.e., we consider the distribution of the number of followers *of the followers*). To account for the cases where an attacker forges this reputation score, we consider the temporal evolution of the reputation score (i.e., the distribution of times at which the identity acquired 0th, 25th, or 50th percentile of their reputation). Since it was easy for us to gather the timestamps at which the identities started building their reputation (i.e., the date at which the identities "joined" the service—corresponding to the 0th percentile of their reputation—we use the join dates of identities to build the reference distribution.

2. Building reference distributions: In Twitter, we assume that the accounts verified by Twitter[4] do not knowingly tamper their follower counts. We use them as the set of known untampered computations. We randomly sample 30,000 verified accounts from the list of Twitter verified accounts with more than 1,000 followers, and crawled profiled information of their 266M followers to derive the reference distribution.

3. Selecting a threshold: We compute KL-divergence values for all the 69,409 identities and estimate a KL-divergence threshold of 7.88 and 5.79 using the follower count reputation score, and join date, respectively.

[4]Twitter vouches for the authenticity of a small portion of all identities (43,901 identities as of April 2013) through an offline verification process.

4. Detecting tampered computations: Table 5.2 shows the number of anomalous computations flagged using follower count and join dates. Among the 69,409 popular users, using follower counts, Stamper flags 620 users and using join dates, Stamper flags 1,129 users as having tampered follower counts. When we examine the overlap between computations flagged using follower count and join date, it is very low, consisting of only 49 computations. Thus, using join dates, Stamper is able to flag 1,080 users (with potentially tampered follower counts) who were not flagged using the follower count reputation score. These 1,080 users were able to successfully hide or evade detection when using the follower count reputation score. This finding further shows the advantages of using unforgeable timestamps to detect tampering. Our discussion above once again demonstrates the ease of deploying Stamper.

#Flagged using follower count reputation score	**#Flagged using join date, i.e., 0% reputation time**	**#ids in common**
620	1,129	49

Table 5.2: Number of anomalies flagged among the 69,409 identities using the follower count reputation score and timestamp at 0th percentile reputation (join date).

5.4.2.2 Investigating anomalies to detect new attacks

For manual investigation, we randomly sample 50 computations out of each group of anomalous computations flagged using follower count and join dates, respectively. Three graduate students with prior experience in investigating suspicious identities in social computing systems spent roughly 15 to 20 minutes per computation for investigation.

First, we try to understand the characteristics of the distribution for each anomalous sample. Second, we attempt to localize our analysis to a subset of the identities within the tampered computation that are most likely to be Sybils. We can find such a subset by looking for regions within the anomalous distribution where it exhibits maximum divergence from the reference distribution. To identify potential Sybil identities, for each candidate account,

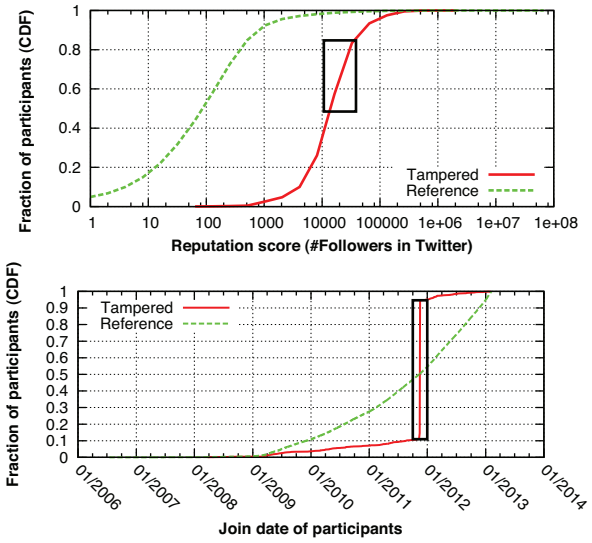


Figure 5.5: Detecting suspicious participants of a tampered computation. The highlighted region contains the identities suspected of tampering the computation.

we analyze the Twitter profile picture, name, bio, content of tweets posted (including URLs posted), follower and following count, and profiles of followers of the account.

Investigating anomalies in follower count distributions: On investigating the distribution of follower counts we noticed two distinct patterns. In the first pattern, we found that most of the followers have very low reputation (e.g., almost all had less than 10 followers). 2 out of 50 computations exhibit this pattern. These followers look like fake accounts (fake looking profile, pictures and tweets talking about following activity) and some were already suspended by Twitter. While we would expect tampering to be carried out by unpopular Sybil identities, we were surprised to find only two such instances. The remaining 48/50 computations showed a different distribution pattern–an example is shown in Figure 5.5 (top figure). When analyzing these computations, we surprisingly discovered that these identities appear to be popular users (i.e., more than 1,000 followers) tampering their follower counts by *colluding* with one another and exchanging links with "you follow me, I follow you" deals. Such activity is referred to as "farming" links on the Twitter network. Link farming [105] is a well-studied problem in Twitter and their defining traits are as follows: they have a large number of followers (more than 1,000), a following per follower ratio in the range [0.9,1.1]

and a majority of their followers satisfy these criteria as well. The identities we investigated match these link farmer traits. Using this definition we analyzed all the followers of each of the 48 computations, and found that all computations had at least 32% of their followers that matched the criteria for link farmers. In fact, for 44 out of 48 cases, a majority of their followers are link farmers.

Our analysis above reveals that even though Stamper has been designed to catch computations tampered by Sybil attacks, it is able to detect a broader category of attacks, including *colluding attacks* by non-Sybil identities. However, we do not claim that our robustness guarantees would extend to such non-Sybil attacks.

Investigating anomalies in join date distributions: We analyzed the distributions of the 50 random computations in this case and noticed that for a majority (33/50) of the computations, a significant fraction of the participating identities are tightly clustered in the time domain. Figure 5.5 (bottom figure) gives an example of such a tampered computation. More specifically, we check if at least 10% (12%) of the followers of the identity joined Twitter within a single day (or week).[5] In contrast, the join dates of identities in the reference (untampered) distribution are spread out over many years. The tight-clustering in the time domain suggests that these identities are possibly created by a Sybil attacker on a single day and then pressed into attack soon after.

To further test our hypothesis, we bought followers for 10 different Twitter identities under our control from 10 different online marketplaces. We discovered these services with a simple keyword search (e.g., "buy Twitter followers") on search engines like Google. In each case, we paid to receive 1,000 followers. When we analyzed the timestamp distribution of followers bought from the black-market for the 10 accounts, we observe that for 9 out of 10 accounts, a vast majority of followers were created on the same day or on a handful

[5]Note that we choose conservative thresholds where we found that over 95% of 69K computations did not exhibit this level of tight clustering in the time domain. To validate our thresholds of 10% followers in a day (and 12% in a week), we monitored accounts that exhibited tight clustering and those that did not, for 6 months. Identities forming clusters had a high Twitter suspension rate of 36% compared to a low suspension rate of 0.38% for the other accounts.

of days. This observation supports our hypothesis that identities discovered in anomalous distributions of join dates are Sybils from whom links have been bought.

We further analyzed the remaining 17 (out of 50) computations that did not exhibit tight clustering at a day or week's granularity. On manual investigation of followers in the most divergent region, we found that 15/17 computations had very suspicious followers and are most likely tampered computations. Many followers look like fake profiles with no profile picture, names with specific patterns, meaningless tweet content, and some even had malware links in their tweets. An interested reader can browse through more details of the manual analysis of these 15 suspicious computations on this page: `http://trulyfollowing.app-ns.mpi-sws.org/local/stamper/tampered_fcounts.html`. Note that our manual investigation provides detailed information about why we think a computation looks suspicious along with a sample of suspicious participants of the computation.

5.4.2.3 Detecting new Sybil identities

We now show how an operator can leverage the newly discovered attacker strategies to detect new Sybil identities. We propose to identify cases of follower tampering by analyzing the join date distributions of followers of a given identity and checking if a non-trivial fraction of their followers joined Twitter within a small window of time (we use the same thresholds discussed in earlier section). We applied this technique to detect potential follower tampering activity in Twitter to over 2.1M identities in Twitter with more than 1000 followers. We detect 89,728 identities as having tampered follower counts. Interested readers can browse the data about these 89,728 identities at the site: `http://trulyfollowing.app-ns.mpi-sws.org/`. From these flagged computations, we identified over 23 million Sybil followers whose join dates fall within a small window of time.[5]

5.4.3 Case 3: Tweet promotion tampering

Goal: Find tampered tweet promotions: Twitter tends to preferentially recommend and rank (while searching) content that has been posted (promoted) by a larger number of users [14]. So popular content, be it tweets, or URLs or keywords (also known as *topics* in Twitter) has a higher chance to become even more popular in Twitter. As a result, there are strong incentives for users to artificially boost popularity of their posts by hiring Sybil identities to promote their content [13]. We are interested in three types of crowd computations involving a set of identities that: (1) retweet a tweet, (2) tweet a URL, or (3) tweet about a particular topic (described by keyword(s)). Our goal is to detect attacks that manipulate such computations, i.e., detect content (tweet, URL or topic) that is promoted by Sybil identities.

Data gathered: The Twitter API streams a small (1%) random sample of all public tweets [27]. We target detecting promotion tampering for content that has been promoted by more than 500 users. To find crowd computations involving retweets and URLs, we use the Twitter streaming API [27] to collect five months (26th February 2014 to 7th August 2014) of public tweets on Twitter (this is a 1% random sample of all tweets). We collected over 328M tweets by over 52M users that were either retweets and/or contained at least one URL. To obtain data about computations involving topics, we collected the globally popular *trending topics*—Twitter recommends a set of 10 globally trending topics every 5 minutes—from 16th July 2014 to 8th September 2014. This included over 23M tweets covering 11,976 trending topics on Twitter. For our analysis, from the above data, we only consider crowd computations for which we could get data for more than 500 participants. Table 5.3 shows the number of computations we consider in each of the three categories (URLs, retweets and trending topics) obtained from the collected data. We succeeded in collecting data for 5,433 URL computations, 1,916 retweet computations and 7,226 trending topic computations.

URL	Retweet	Trending topic
509 / 5,433	38 / 1,916	157 / 7,226

Table 5.3: Number of anomalies flagged and the total number of computations in each content category.

5.4.3.1 Ease of deploying Stamper

1. Creating a pool of reputation scores: Here we focus on the temporal evolution of reputation score. Similar to the follower tampering case study, we use the join dates (timestamp at 0th percentile reputation) of identities to build the reference distribution.

2. Building reference distributions: Similar to the previous case study, we assume that Twitter verified accounts do not knowingly tamper tweet promotion. We use the set of tweets posted by verified users as known untampered computations. More specifically, for a tweet posted by a verified user, we form a crowd computation by considering the set of users who retweet the tweet. We randomly sample 10 tweets out of all tweets by each verified user among the most popular 2500 verified users (based on follower count). We were able to gather data for 16,304 untampered crowd computations (having more than 500 participants) involving 10,333,036 unique participants.

3. Selecting a threshold: We compute KL-divergence values for all 1,916 retweet computations and obtain a KL-divergence threshold of 12.13.

4. Detecting anomalous computations: Using the reference distribution and inferred threshold, we apply Stamper to detect anomalous computations. Table 5.3 shows the number of anomalous computations flagged for the different types of content. Stamper flagged 704 computations out of a total of 14,575 computations across all three content categories. It is interesting to note that there are anomalies in all content categories, including the trending topic category, which are topics recommended by Twitter on the front page as trending worldwide!

5.4.3.2 Investigating anomalies

We randomly sample 50 flagged computations (all 38 in the case of retweets) for each type of content (total of 138 computations) for investigation. We follow the methodology discussed earlier in Section 5.4.2.2. First, we observe that a significant fraction of identities in many computations are tightly clustered in the time domain (similar to what we saw in Figure 5.5) in contrast to the identities in the reference distribution whose join dates are spread over many years. In all three content categories, a majority of computations had at least 10% of their participants who joined Twitter on the same day. This clustering in join dates suggests that attackers are creating accounts in bulk in a short span of time and using them for attacks.

Next, when we manually investigate the flagged computations, we find that 49/50, 38/38, and 50/50 computations in URL, retweet and trending topic categories, respectively, have participating identities that exhibit suspicious behavior. For example, in the trending topic computations, we find participant identities that look very similar to each other in terms of profile features and tweeting patterns and promote black-market service URLs by piggybacking on trending topic keywords. By doing so, the attacker gets much higher exposure for their content because trending topics are recommended to all users by Twitter. Interested readers can findings more details about our manual investigation at the following web page: `http://trulyfollowing.app-ns.mpi-sws.org/local/stamper/tampered_tweets.html`, which lists the details of all the 137 suspicious computations and explanations for why we find them suspicious.

5.4.3.3 Stamper deployment

Finally, to demonstrate the effectiveness of Stamper in the real world, we deployed a public online service at `http://trulytweeting.app-ns.mpi-sws.org/` which detects *tampered tweet promotions* in Twitter. Our service lists currently *trending topics*, popular URLs and tweets that are tampered and also provides a real time search interface

to check arbitrary URL or topic computations for tampering in Twitter. We encourage interested readers to test the service to understand the potential and practicality of Stamper.

5.4.4 Ethics

All the data about user activity collected from Yelp and Twitter are publicly visible information. All money we paid to acquire followers from the black-market were exclusively for Twitter accounts under our control and setup for the sole purpose of conducting the experiments in this chapter. Overall, we ensured that no user or page on Yelp and Twitter was abused or benefited as a result of our study.

5.5 Limitations and future work

Limitation 1: Can Stamper detect attacks using non-Sybil identities that are incentivized to collude or whose login credentials are compromised? Stamper cannot provide the robustness guarantees discussed in Section 5.2.3, for attacks involving non-Sybil identities that are compromised by an attacker or that have an incentive to collude with one another (e.g., to boost each other's popularity). However, there is still hope; Stamper would still be able to detect tampering as long as the colluding or compromised identities are not carefully chosen in such a way that the distribution of reputation scores and their temporal evolution match that of non-Sybil identities. In practice, it may not be easy for an attacker to selectively target and compromise non-Sybil identities with varied levels of reputation scores. In fact, in our evaluation Section 5.4.2, we show that Stamper is able to detect identities colluding to follow one another in Twitter, because their collusion distorts the distribution of their reputation scores, which is easily flagged by Stamper.

Limitation 2: Can Stamper detect computations that involve only a few identities or that have been tampered using only a few Sybils? At its core, Stamper relies on identifying *statistically significant* deviations in distributions of reputation histories of

identities participating in a crowd computation. So the robustness guarantees of Stamper do not hold when the number of participating identities is too small or when the degree of tampering is small. In practice, we show that Stamper can be used to detect tampering of computations with 100 or more participants (see Section 5.4.1). We also show that Stamper is very robust in detecting highly tampered computations (e.g., > 50% of identities are Sybil), but when the computations are tampered only to a small extent (e.g., < 10% of identities are Sybil), the detection accuracy suffers. While this is a fundamental limitation of Stamper's approach, it is worth noting that in practice, system operators would be more concerned about detecting heavily tampered computations than lightly tampered ones.

Future work: Note that currently Stamper's primary goal is to detect the presence of tampering. As part of future work, we plan to extend Stamper to enable Sybil resilient crowd computations. We plan to explore techniques to automatically filter out malicious identities in a tampered crowd computation. One approach would be to automatically remove portions of the distribution (the anomalous distribution) that exhibits maximum divergence from the reference distribution, such that the target distribution better aligns (i.e, has lower divergence) with the reference distribution. However, such a technique has to be calibrated carefully such that honest crowd participants are not removed (low false positives) and most of the malicious identities are removed (high true positives).

5.6 Conclusion

In this chapter, we tackle the challenging problem of detecting when computations on crowdsourcing systems like Twitter or Yelp have been tampered by fake (Sybil) identities. We have advocated a fundamentally different approach called Stamper that can detect whether a computation has been tampered even when it is not feasible to detect which of the individual identities participating in the computation are Sybil. The key insight that enables our approach is that large statistical samples (groups) of Sybil and non-Sybil

identities exhibit very different characteristics. We have leveraged this insight to design Stamper to be (i) capable of detecting tampered computations and raising the bar for defense against adaptive attackers and (ii) capable of detecting computation tampering independent of the attacker strategy. We have demonstrated the robustness and practicality of Stamper by evaluating its performance using extensive data gathered from two widely used crowd-sourcing systems, namely Yelp and Twitter.

CHAPTER 6

Conclusion

Today, social computing systems are used by more than a billion people for a variety of different online activities, such as connecting and socializing with friends/family or people with other similar interests, learning, trading goods, and playing games. Unfortunately, service abuse is a serious problem affecting the quality of service experienced by users of these platforms and also the sustainability of these sites. Since attackers of these platforms are mostly driven by goals of financial gain, we focus on techniques that are *robust*, such that the attacker is disincentivized from an economic viewpoint. Further, given that most social computing platforms today rely on a weak identity infrastructure (which is easily exploited by attackers), we focus on defenses that can be *practically* deployed today without any change to the underlying identity infrastructure. In this dissertation, we took a few steps based on the above two goals and presented new schemes to mitigate service abuse in social computing systems.

We started by analyzing existing social network-based Sybil detection schemes to better understand how they work and their practical limitations when applied to real world social networks. We found that even though different schemes use different types of graph analysis algorithms, at their core, these schemes are essentially finding communities in the social graph. This has two main implications. First, existing schemes can misclassify honest users as Sybils because honest users tend to form communities in real world social networks and the assumption about the honest region of the network being fast mixing may not hold true in practice. Second, researchers now have an opportunity to leverage the vast

literature of network community detection algorithms to improve social network-based Sybil detection schemes. In fact, recent work [49, 201, 174, 48] has leveraged findings from our study [196] to propose improved Sybil detection schemes. Finally, as the name suggests, these schemes are best suited for detecting Sybil (or fake) identities in the network and provide no guarantees for detecting compromised or colluding identities. However, the biggest advantage with this approach is that it is easy to integrate with any social computing application and the only requirement is the presence of a social network (that satisfies certain assumptions discussed in Chapter 2).

Next, we presented a different approach to social network-based Sybil defense called Sybil tolerance. Sybil tolerance schemes stand out from all the other techniques presented in this thesis, by not focussing on detecting attacks (or identities involved in the attack), but instead, directly limiting the impact that Sybil identities can have on non-Sybil identities in the system. This allows operators to design applications that have built-in resilience to Sybil attacks. We presented a general methodology for designing Sybil tolerance schemes using credit networks and showed how existing Sybil tolerance approaches work by conducting payments over credit networks. Unfortunately, these schemes do not scale well to large social networks because credit payments over large social networks is computationally expensive. To get around this practical deployment barrier, we designed and implemented Canal, a system that can efficiently approximate credit payments over large, dynamic networks. We showed that Canal can be integrated into existing Sybil tolerance schemes and provide several orders of magnitude speed-up in credit payment calculations. However, it should be noted that unlike social network-based Sybil detection schemes, Sybil tolerance schemes need to be deeply integrated into the operation of an application and thus needs to be tailored for each application. In terms of security guarantees, Sybil tolerance schemes provide no robust guarantees against attacks using compromised or colluding identities (similar to Sybil detection approaches). However, in our later work using Canal [154], we showed that it is possible to build a Sybil tolerance scheme that can defend against

attacks using compromised identities by using appropriate payment mechanisms, as long as the attacker has access to a limited set of compromised identities and is not capable of selectively targeting and compromising identities in the network.

Not all social computing systems have a social network (and one that satisfies certain assumptions required by social network-based defenses). To build defenses that can be applied more generally, in our next two approaches, we leveraged activity or behavioral history of identities (e.g., based on how an identity interacts with content or other users in the system). We proposed two new approaches based on anomaly detection on user social behavior.

Our first approach based on behavioral profiling, focused on identifying individual misbehaving identities by analyzing their social behavior for anomalous behavioral patterns. Our idea is based on using Principal Component Analysis to learn only normal patterns of user social behavior, and flag any behavior that deviates significantly from normal as anomalous. Such an approach has the potential to defend against adaptive attackers (or attackers who mutate and change strategy to adapt to a particular defense), as we do not make any assumptions about the attacker's strategy. Our PCA-based anomaly detection technique, when applied to diverse real world attack data involving Sybil, compromised and colluding identities, identified misbehaving users with high accuracy. Finally, we applied our technique to detect click-spam on Facebook ads and surprisingly identified that a significant fraction of clicks from our ad campaigns looked anomalous. Our technique could be used by advertisers, ad agencies, or concerned third-parties to apply to their own ad campaigns to estimate the level of anomalous traffic. While our approach can detect a wide variety of attacks, it would be hard to defend against an attacker who uses newly created Sybil accounts (with no activity history) or Sybil accounts with very limited activity history. Our PCA-based tool would fail to find any anomalous patterns when there is no activity data or very minimal activity information. The same limitation would also apply to newly compromised identities or colluding identities with very limited activity history. This

limitation motivated us to explore a new approach that focused on identifying a group of misbehaving identities instead of individual identities.

Our second approach, called Stamper, is based on the idea that even when it is fundamentally hard to distinguish between individual Sybil and non-Sybil identities (e.g., when both have no activity history), large groups of Sybil and non-Sybil identities can be differentiated. Further, our work is inspired by the trend where popular social computing systems are increasingly employing crowd computing (polling the opinions of a crowd or a group of users) to rate/rank content, users or businesses. Unfortunately, crowd computations today are tampered by Sybil attackers with the goal of manipulating crowd opinion. Stamper is designed to detect tampered crowd computations where a significant fraction of the participants of the crowd are Sybil identities. Stamper looks for anomalous patterns in crowd behavior (instead of individual user behavior) by leveraging features that are unforgeable, thus essentially raising the bar for evasion by adaptive attackers. We evaluated Stamper on data gathered from two popular social computing systems, Twitter and Yelp and discovered thousands of previously unknown tampered computations. While Stamper provides robust security guarantees against Sybil attacks, the same guarantees do not extend to cases where attackers use compromised or colluding identities. However, we show that Stamper is still useful in scenarios where the attacker does not have the ability to carefully sample colluding or compromised identities in a specific manner from the user population.

Lastly, we note that schemes like Stamper and Sybil tolerance enable new approaches to securing today's online services. On the other hand, our PCA-based anomaly detection approach is a significant improvement over well studied existing approaches that focus on detecting individual misbehaving identities. Further, our PCA-based scheme can complement schemes like Stamper to defend against future manipulations. Stamper does not identify the individual malicious participants of the tampered crowd. This would enable an attacker to repeatedly use the same malicious identities for future manipulations or other types of misbehavior. To limit such repeated attacks, the operator could use our PCA-based

scheme to remove malicious participants (except in cases where they are fundamentally hard to detect) from the tampered crowd.

Appendices

APPENDIX A

Analysis of Sybil detection schemes

A.1 Analysis of SybilGuard

Assumed social network topology: SybilGuard [210] assumes that the non-Sybil region is fast mixing [151], meaning that after $O(\log n)$ hops (where n is the number of non-Sybils), the probability distribution of the last node on a random walk reaches the stationary distribution. SybilGuard assumes that the entire network (the Sybil region combined with the non-Sybil region) is not fast mixing.

Partitioning algorithm: SybilGuard uses constrained random walk for marking nodes as non-Sybil or Sybil. It marks a suspect node as non-Sybil if the random walk from the trusted node and the suspect intersect, otherwise the suspect is marked as a Sybil.

Node ranking by partitioning algorithm: In order to generate a ranking, we conduct random walks from the trusted node. We start with a walk length 1 and increase it to k, where k is the length of the random walk such that all nodes in the network are marked as non-Sybil. The order in which nodes are marked as non-Sybil in these increasingly long random walks imposes a ranking. In the rare case when all the nodes in the network are not marked as non-Sybil using a single random seed and a long walk length, we conduct a series of random walks with different random seeds to induce a ranking for the remaining nodes.

Determining cutoff: SybilGuard uses $O(\sqrt{n}\log n)$ random walks to gather samples from the non-Sybil region of n nodes. For a social network with $O(\log n)$ mixing time, based on the birthday paradox, two non-Sybil nodes with $\sqrt{n}$ samples from the non-Sybil region will

have an intersection with high probability. SybilGuard relies on an estimation procedure for determining the appropriate length of the random walk, and consequently, the cutoff value.

A.2 Analysis of SybilLimit

Assumed social network topology: SybilLimit [209] makes the same assumptions about the network as SybilGuard.

Partitioning algorithm: SybilLimit performs $O(\sqrt{m})$ independent random walks of length $O(\log n)$ from each node. Two conditions must be satisfied for the trusted node to mark a suspect as a non-Sybil. The first condition—called the *intersection condition*—requires that the last edge of one of the random walks of the trusted node and the suspect must intersect. The second condition—called the *balance condition*—limits the number of non-Sybils per attack edge. Each tail of a random walk is assigned a "load" that is not allowed to exceed a given threshold; the load is incremented each time the trusted node marks another suspect as a non-Sybil.

Node ranking by partitioning algorithm: SybilLimit has two primary parameters for controlling the number of nodes marked as non-Sybil in the network—the number of random walks from each node and the length of these walks. As these parameters are increased, greater numbers of nodes are marked as non-Sybil. Similar to SybilGuard, we infer a ranking based on the order in which nodes are marked as non-Sybil.

Determining cutoff: Similar to SybilGuard, SybilLimit relies on an estimation procedure to find length of random walk and the number of random walk required. These two parameters impose a cutoff.

A.3 Analysis of SybilInfer

Assumed social network topology: SybilInfer [65] makes the same assumption as SybilGuard. SybilInfer also makes a further assumption that the modified random walks are fast mixing in real social networks.

Partitioning algorithm: SybilInfer performs multiple random walks from each node to sample nodes from the non-Sybil region. It further uses a Bayesian inference technique to determine the probability of any node in the system being marked as non-Sybil.

Node ranking by partitioning algorithm: Since SybilInfer assigns each node a probability of being a non-Sybil, the nodes can be ranked based on this probability. We conduct 30 runs of SybilInfer with different random seeds, and use the average probability over all the runs to determine the final ranking of the nodes.

Determining cutoff: SybilInfer partitions the nodes based on a threshold value for the probability of a node being non-Sybil.

A.4 Analysis of SumUp

Assumed social network topology: SumUp assumes that the min-cut between the vote collector (i.e., the trusted node) and non-Sybil nodes occurs at the collector, and that the min-cut between Sybils and the non-Sybils occurs at the attack edges.

Partitioning algorithm: SumUp partitions nodes based on whether their vote is accepted or not. Nodes whose votes are accepted are treated as non-Sybils, whereas nodes whose votes are subject to capacity constraints are treated as Sybils.

Node ranking by partitioning algorithm: SumUp decides whether a vote will be collected or not by defining a *voting envelope* within which all votes are collected and outside of which votes are constrained to one per link out of the envelope. The size of the voting envelope is controlled by the parameter C_{max}, which is the maximum number of votes that

can be collected by the trusted node. In order to rank nodes, we increase C_{max} from 1 to k, where k is the value for which the voting envelope contains the entire network. The order in which these nodes are added to the voting envelope induces a ranking.

Determining cutoff: C_{max} determines the size of the voting envelope and serves as the cut-off parameter.

BIBLIOGRAPHY

[1] A Rave, a Pan, or Just a Fake? http://tinyurl.com/nyt-haggl.

[2] After Sting Operation, Yelp Outs 8 Businesses That It Caught Trying To Buy Reviews. http://tinyurl.com/yelp-suspend.

[3] AOL Instant Messenger. https://www.aim.com/.

[4] Bulletin board system. https://en.wikipedia.org/wiki/Bulletin_board_system.

[5] Buy Reviews on Yelp, Get Black Mark. http://www.nytimes.com/2012/10/18/technology/yelp-tries-to-halt-deceptive-reviews.html?_r=0.

[6] Buying Their Way to Twitter Fame. http://www.nytimes.com/2012/08/23/fashion/twitter-followers-for-sale.html?_r=0.

[7] Buying Their Way to Twitter Fame. http://tinyurl.com/nyt-tw-sale.

[8] Buying Yelp Reviews. http://tinyurl.com/yelp-bought.

[9] Cyworld. www.cyworld.kr/.

[10] Death by captcha. http://www.deathbycaptcha.com/.

[11] Facebook advertising. https://www.facebook.com/business/products/ads.

[12] Facebook newsfeed. https://www.facebook.com/help/327131014036297/.

[13] Fake Twitter Followers Become Multimillion-Dollar Business. http://tinyurl.com/nytimes-fake-follower.

[14] FAQs about trends on Twitter. http://tinyurl.com/twitter-faq-trends.

[15] Google+ platform for web. https://developers.google.com/+/web/?hl=en.

[16] Harvard study: Yelp drives demand for independent restaurants. http://officialblog.yelp.com/2011/10/harvard-study-yelp-drives-demand-for-independent-restaurants.html.

[17] How low-paid workers at 'click farms' create appearance of online popularity. http://tinyurl.com/guardian-cf-p.

[18] Internet relay chat. https://en.wikipedia.org/wiki/Internet_Relay_Chat.

[19] Like Button for the Web. https://developers.facebook.com/docs/reference/plugins/like/.

[20] Linkedin plugins. https://developer.linkedin.com/plugins.

[21] Marketing on Facebook starts with a Page. https://www.facebook.com/about/pages.

[22] Microsoft warns of new Trojan hijacking Facebook accounts. http://news.cnet.com/8301-1009_3-57584111-83/microsoft-warns-of-new-trojan-hijacking-facebook-accounts/.

[23] Sixdegrees.com. https://en.wikipedia.org/wiki/SixDegrees.com.

[24] Statistical distance. http://en.wikipedia.org/wiki/Statistical_distance.

[25] Twitter for websites. https://dev.twitter.com/web/overview.

[26] Twitter now has 75M users; most asleep at the mouse. http://tinyurl.com/twitter-inactive.

[27] Twitter Public API: one percent sample. http://tinyurl.com/tw-rand-1.

[28] Value of a facebook like. http://mashable.com/2013/04/17/facebook-fan-value-researcher/.

[29] Web browser automation using selenium. http://www.seleniumhq.org/.

[30] Why Yelp has a Review Filter. http://officialblog.yelp.com/2009/10/why-yelp-has-a-review-filter.html.

[31] Yahoo! GeoCities. https://en.wikipedia.org/wiki/Yahoo!_GeoCities.

[32] Yelp Consumer Alerts: Letting You Know Before You Spend Your Dough. http://tinyurl.com/yelp-consumer-alert.

[33] Yelp's Review Filter Explained. http://officialblog.yelp.com/2010/03/yelp-review-filter-explained.html.

[34] Youtube statistics. https://www.youtube.com/yt/press/statistics.html.

[35] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *ACM Transactions on Internet Technology (TOIT)*, 5(2):299–327, 2005.

[36] Advogato trust network. http://www.trustlet.org/wiki/Advogato.

[37] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, 2007.

[38] R. Andersen and K. J. Lang. Communities from seed sets. In *Proceedings of the 15th International Conference on World Wide Web (WWW)*, 2006.

[39] Apache Mahout. `http://mahout.apache.org/`.

[40] J. P. Bagrow. Evaluating local community methods in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(05):P05001, 2008.

[41] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[42] Facebook Advertisement Value (BBC). `http://www.bbc.co.uk/news/technology-18813237`.

[43] G. Begelman, P. Keller, F. Smadja, et al. Automated tag clustering: Improving search and exploration in the tag space. In *Proceedings of the Collaborative Web Tagging Workshop at WWW'06*, 2006.

[44] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida. Detecting spammers on Twitter. In *Proceedings of the 7th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*, 2010.

[45] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. CopyCatch: Stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web (WWW)*, 2013.

[46] P. Bhattacharya, S. Ghosh, J. Kulshrestha, M. Mondal, M. B. Zafar, N. Ganguly, and K. P. Gummadi. Deep Twitter diving: Exploring topical groups in microblogs at scale.

In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing (CSCW)*, 2014.

[47] N. Borisov. Computational puzzles as Sybil defenses. In *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P)*, 2006.

[48] Y. Boshmaf, K. Beznosov, and M. Ripeanu. Graph-based Sybil detection in social and information systems. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2013.

[49] Z. Cai and C. Jermaine. The latent community model for detecting Sybil attacks in social networks. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS)*, 2012.

[50] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI)*, 2012.

[51] Q. Cao, X. Yang, J. Yu, and C. Palow. Uncovering large groups of active malicious accounts in online social networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[52] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *SIGOPS Operating Systems Review*, volume 36, pages 299–314, 2002.

[53] M. Cataldi, L. Di Caro, and C. Schifanella. Emerging topic detection on Twitter based on temporal and social terms evaluation. In *Proceedings of the Tenth International Workshop on Multimedia Data Mining (MDMKDD)*, 2010.

[54] R. B. Cattell. The scree test for the number of factors. *Multivariate behavioral research*, 1(2):245–276, 1966.

[55] M. Cha, F. Benevenuto, H. Haddadi, and K. Gummadi. The world of connections and information flow in twitter. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 42(4):991–998, 2012.

[56] M. Cha, H. Haddadi, F. Benevenuto, and P. K. Gummadi. Measuring user influence in twitter: The million follower fallacy. In *Proceedings of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM)*, 2010.

[57] H. Chun, H. Kwak, Y.-H. Eom, Y.-Y. Ahn, S. Moon, and H. Jeong. Comparison of online social relations in volume vs interaction: A case study of Cyworld. In *Proceedings of the 8th ACM/USENIX Internet Measurement Conference (IMC)*, 2008.

[58] A. Clauset. Finding local community structure in networks. *Physical review E*, 72(2):026132, 2005.

[59] R. T. Clemen and R. L. Winkler. Combining probability distributions from experts in risk analysis. *Risk analysis*, 19(2):187–203, 1999.

[60] Click Farms (Washington Post). `http://www.washingtonpost.com/blogs/wonkblog/wp/2014/01/06/click-farms-are-the-new-sweatshops`.

[61] Collusion Network Site #1. `http://addmefast.com/`.

[62] Collusion Network Site #2. `http://likesasap.com/`.

[63] C. Croux, P. Filzmoser, and M. R. Oliveira. Algorithms for projection–pursuit robust principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 87(2):218–225, 2007.

[64] P. Dandekar, A. Goel, R. Govindan, and I. Post. Liquidity in credit networks: A little trust goes a long way. In *Proceedings of the 12th ACM Conference on Electronic Commerce (EC)*, 2011.

[65] G. Danezis and P. Mittal. SybilInfer: Detecting Sybil nodes using social networks. In *Proceedings of the 16th Network and Distributed System Security Symposium (NDSS)*, 2009.

[66] A. Das Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the 3rd ACM International Conference of Web Search and Data Mining (WSDM)*, 2010.

[67] V. Dave, S. Guha, and Y. Zhang. Measuring and fingerprinting click-spam in ad networks. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2012.

[68] V. Dave, S. Guha, and Y. Zhang. Viceroi: Catching click-spam in search ad networks. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, 2013.

[69] Q. Ding and E. D. Kolaczyk. A compressed PCA subspace method for anomaly detection in high-dimensional data. *IEEE Transactions on Information Theory*, 59(11):7419–7433, 2013.

[70] E. Dinic. An algorithm for the solution of the max-flow problem with the polynomial estimation. *Doklady Akademii Nauk SSSR*, 194(4):1277–1280.

[71] D. do B. DeFigueiredo and E. T. Barr. Trustdavis: A non-exploitable online reputation system. In *Proceedings of the 7th IEEE International Conference on E-Commerce Technology (IEEE E-Commerce)*, 2005.

[72] J. Douceur. The Sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[73] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna. Compa: Detecting compromised accounts on social networks. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS)*. The Internet Society, 2013.

[74] Facebook usage statistics. http://newsroom.fb.com/company-info/.

[75] Facebook Accounts Compromised. http://mashable.com/2013/12/04/hackers-facebook-twitter-gmail/.

[76] Facebook Ads Fake Clicks. http://www.businessinsider.com/mans-600000-facebook-ad-disaster-2014-2.

[77] Facebook Advertisement Value (Salon). http://www.salon.com/2014/02/14/facebooks_big_like_problem_major_money_and_major_scams/.

[78] Facebook Black Market Service #1. http://fbviro.com.

[79] Facebook Black Market Service #2. http://get-likes.com/facebook-likes-store/.

[80] Facebook Black Market Service #3. http://www.buyfbservices.com.

[81] Facebook Black Market Service #4. http://twittertechnology.com/facebook.

[82] Facebook Black Market Service #5. http://www.pagelution.com/buy-facebook-fans.html.

[83] Facebook Black Market Service #6. http://teamfollowpromo.com/facebooklikes.

[84] Facebook COSN Group. https://www.facebook.com/groups/cosn2013.

[85] Facebook Directory. https://www.facebook.com/directory/people/.

[86] Facebook Graph Search. https://www.facebook.com/about/graphsearch.

[87] Fake Likes on Facbook and Instagram. http://www.huffingtonpost.com/2013/08/16/fake-instagram-likes_n_3769247.html.

[88] Facebook Malware Extension. http://bit.ly/1mIum7L.

[89] Facebook Internet Explorer Malware. http://blogs.technet.com/b/mmpc/archive/2013/11/14/febipos-for-internet-explorer.aspx.

[90] Facebook Page insights. https://www.facebook.com/help/336893449723054.

[91] Facebook Quarterly Report (2013). http://investor.fb.com/releasedetail.cfm?ReleaseID=821954.

[92] Facebook's estimate of fraction of undesirable accounts for 2013. http://investor.fb.com/.

[93] Facebook SIGCOMM Group. https://www.facebook.com/groups/sigcomm/.

[94] Facebook Timeline. https://www.facebook.com/about/timeline.

[95] Febipos.A Malware. http://www.microsoft.com/security/portal/threat/encyclopedia/Entry.aspx?Name=Trojan:JS/Febipos.A.

[96] S. Feng, L. Xing, A. Gogar, and Y. Choi. Distributional footprints of deceptive product reviews. In *Proceedings of the the 6th International AAAI Conference on Weblogs and Social Media (ICWSM)*, 2012.

[97] J. Fogarty, R. S. Baker, and S. E. Hudson. Case studies in the use of ROC curve analysis for sensor-based estimates in human computer interaction. In *Proceedings of Graphics Interface 2005*, 2005.

[98] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

[99] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[100] Friendster. `http://www.friendster.com`.

[101] M. Gabielkov and A. Legout. The complete picture of the twitter social graph. In *Proceedings of the 2012 ACM Conference on CoNEXT student workshop*, 2012.

[102] H. Gao, Y. Chen, K. Lee, D. Palsetia, and A. Choudhary. Towards online spam filtering in social networks. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS)*, 2012.

[103] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao. Detecting and characterizing social spam campaigns. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC)*, 2010.

[104] A. Ghosh, M. Mahdian, D. M. Reeves, D. M. Pennock, and R. Fugger. Mechanism design on trust networks. In *Proceedings of the 3rd International Conference on Internet and Network Economics (WINE)*, 2007.

[105] S. Ghosh, B. Viswanath, F. Kooti, N. K. Sharma, G. Korlam, F. Benevenuto, N. Ganguly, and K. P. Gummadi. Understanding and combating link farming in the twitter

social network. In *Proceedings of the 21st International Conference on World Wide Web (WWW)*, 2012.

[106] O. Goga, G. Venkatadri, and K. P. Gummadi. The doppelgänger bot attack: Exploring identity impersonation in online social networks. In *Proceedings of the 15th ACM/USENIX Internet Measurement Conference (IMC)*, 2015.

[107] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. In *Proceedings of the 18th annual ACM Symposium on Theory of Computing (STOC)*, 1986.

[108] R. E. Gomory and T. Hu. Multi-terminal network flows. *SIAM*, 9(4):551–570, 1961.

[109] The Plus in Google Plus? It's Mostly for Google. `http://www.nytimes.com/2014/02/15/technology/the-plus-in-google-plus-its-mostly-for-google.html`.

[110] Google+ usage statistics. `https://en.wikipedia.org/wiki/Google%2B`.

[111] D. Gregor and A. Lumsdaine. The parallel BGL: A generic library for distributed graph computations. In *Proceedings of the Parallel Object-Oriented Scientific Computing (POOSC)*, 2005.

[112] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @ spam: the underground on 140 characters or less. In *Proceedings of the 17th ACM conference on Computer and communications security (CCS)*, 2010.

[113] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proceedings of the 19th ACM international conference on Information and knowledge management (CIKM)*, 2010.

[114] R. Guimera, L. Danon, A. Diaz-Guilera, F. Giralt, and A. Arenas. Self-similar community structure in a network of human interactions. *Physical review E*, 68(6):065103, 2003.

[115] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. WTF: The who to follow service at Twitter. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, 2013.

[116] H. Haddadi. Fighting online click-fraud using bluff ads. *ACM SIGCOMM Computer Communication Review*, 40(2):21–25, 2010.

[117] T. Hartmann and D. Wagner. Fully-dynamic cut tree construction. Technical Report 2011.25, Karlsruhe Institute of Technology, 2011.

[118] V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[119] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 2011.

[120] J. Jiang, C. Wilson, X. Wang, P. Huang, W. Sha, Y. Dai, and B. Y. Zhao. Understanding latent interactions in online social networks. In *Proceedings of the 10th ACM/USENIX Internet Measurement Conference (IMC)*, 2010.

[121] N. Jindal and B. Liu. Opinion Spam and Analysis. In *Proceedings of the 1st ACM International Conference on Web Search and Data Mining (WSDM)*, 2008.

[122] K. Lee and J. Caverlee and and S. Webb. Uncovering social spammers: Social honeypots + machine learning. In *Proceedings of the 33rd Annual ACM SIGIR Conference (SIGIR)*, 2010.

[123] A. M. Kakhki, C. Kliman-Silver, and A. Mislove. Iolaus: Securing online content rating systems. In *Proceedings of the 22nd International World Wide Web Conference (WWW)*, 2013.

[124] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International Conference on World Wide Web (WWW)*, 2003.

[125] C. Kanich, N. Weaver, D. McCoy, T. Halvorson, C. Kreibich, K. Levchenko, V. Paxson, G. M. Voelker, and S. Savage. Show me the money: Characterizing spam-advertised revenue. In *Proceedings of the 20th USENIX conference on Security (Usenix Security)*, 2011.

[126] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *Proceedings of the 23rd USENIX Security Symposium (Usenix Security)*, 2014.

[127] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the 32nd annual ACM symposium on Theory of computing (STOC)*, 2000.

[128] Y. Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.

[129] B. Krishnamurthy and C. E. Wills. On the leakage of personally identifiable information via online social networks. In *Proceedings of the 2nd ACM workshop on Online social networks (WOSN)*, 2009.

[130] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[131] L2 Norm. `http://mathworld.wolfram.com/L2-Norm.html`.

[132] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2004.

[133] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2010.

[134] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[135] J. Leskovec, K. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th International World Wide Web Conference (WWW)*, 2010.

[136] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International World Wide Web Conference (WWW)*, 2008.

[137] C. Lesniewski-Laas. A Sybil-proof one-hop DHT. In *Proceedings of the 1st Workshop on Social Network Systems*, 2008.

[138] C. Lesniewski-Laas and M. F. Kaashoek. Whānau: A Sybil-proof distributed hash table. In *Proceedings of the 7th Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[139] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina. Detection and identification of network anomalies using sketch subspaces. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC)*, 2006.

[140] E. Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, 2013.

[141] E.-P. Lim, V.-A. Nguyen, N. Jindal, B. Liu, and H. W. Lauw. Detecting product review spammers using rating behaviors. In *Proceedings of the 19th ACM international conference on Information and knowledge management (CIKM)*, 2010.

[142] LinkedIn. `http://www.linkedin.com`.

[143] D. Liu, X.-S. Hua, L. Yang, M. Wang, and H.-J. Zhang. Tag ranking. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*, 2009.

[144] F. Luo, J. Z. Wang, and E. Promislow. Exploring local community structures in large networks. *Web Intelligent and Agent Systems*, 6(4):387–400, 2008.

[145] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2010.

[146] Zeus Botnet Cleanup (Microsoft). `http://blogs.technet.com/b/microsoft_blog/archive/2014/06/02/microsoft-helps-fbi-in-gameover-zeus-botnet-cleanup.aspx`.

[147] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Growth of the Flickr social network. In *Proceedings of the 1st ACM SIGCOMM Workshop on Social Networks (WOSN)*, 2008.

[148] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM/USENIX Internet Measurement Conference (IMC)*, 2007.

[149] A. Mislove, A. Post, K. P. Gummadi, and P. Druschel. Ostra: Leveraging trust to thwart unwanted communication. In *Proceedings of the 5th Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[150] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel. You are who you know: inferring user profiles in online social networks. In *Proceedings of the 3th ACM international conference on Web search and data mining (WSDM)*, 2010.

[151] M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, Cambridge, UK, 2005.

[152] A. Mohaisen, N. Hopper, and Y. Kim. Keep your friends close: Incorporating trust into social network-based sybil defenses. In *Proceedings of the 30th IEEE Conference on Computer Communications (INFOCOM)*, 2011.

[153] A. Mohaisen, A. Yun, and Y. Kim. Measuring the mixing time of social graphs. In *Proceedings of the 10th ACM/USENIX Internet Measurement Conference (IMC)*, 2010.

[154] M. Mondal, B. Viswanath, A. Clement, P. Druschel, K. P. Gummadi, A. Mislove, and A. Post. Defending against large-scale crawls in online social networks. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2012.

[155] M. Motoyama, D. McCoy, K. Levchenko, S. Savage, and G. M. Voelker. Dirty jobs: The role of freelance labor in web service abuse. In *Proceedings of the 20th USENIX conference on Security (Usenix Security)*, 2011.

[156] A. Mukherjee, B. Liu, and N. Glance. Spotting fake reviewer groups in consumer reviews. In *Proceedings of the 21st International conference on World Wide Web (WWW)*, 2012.

[157] MySpace. http://www.myspace.com.

[158] S. Nagaraja. Anonymity in the wild: Mixes on unstructured networks. In *Proceedings of the 7th Workshop on Privacy Enhancing Technologies (PET)*, 2007.

[159] M. E. Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

[160] M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences (PNAS)*, 98(2):404–409, 2001.

[161] Nist/sematech e-handbook of statistical methods. http://www.itl.nist.gov/div898/handbook/.

[162] M. Ott, Y. Choi, C. Cardie, and J. T. Hancock. Finding deceptive opinion spam by any stretch of the imagination. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, 2011.

[163] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

[164] J. X. Parreira, D. Donato, C. Castillo, and G. Weikum. Computing trusted authority scores in peer-to-peer web search networks. In *Proceedings of the 3rd International workshop on Adversarial information retrieval on the web*, 2007.

[165] A. Post, V. Shah, and A. Mislove. Bazaar: Strengthening user reputations in online marketplaces. In *Proceedings of the 8th Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[166] D. Quercia and S. Hailes. Sybil attacks against mobile users: Friends and foes to the rescue. In *Proceedings of the 29th Conference on Information Communications (INFOCOM)*, 2010.

[167] M. S. Rahman, T.-K. Huang, H. V. Madhyastha, and M. Faloutsos. Efficient and scalable socware detection in online social networks. In *Proceedings of the 21st USENIX Security Symposium (Usenix Security)*, 2012.

[168] H. Ringberg, A. Soule, J. Rexford, and C. Diot. Sensitivity of PCA for traffic anomaly detection. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007.

[169] B. C. Roy. Bounds on the expected entropy and KL-divergence of sampled multinomial distributions. Technical report, MIT Media Lab, June 2011.

[170] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. Tygar. Stealthy poisoning attacks on pca-based anomaly detectors. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):73–74, 2009.

[171] B. I. P. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, N. Taft, and D. Tygar. Compromising PCA-based anomaly detectors for network-wide traffic. Technical report, EECS Department, University of California, Berkeley, 2008.

[172] S. Seuken and D. C. Parkes. On the Sybil-proofness of accounting mechanisms. In *Proceedings of the 6th Workshop on the Economics of Networks, Systems and Computation (NetEcon)*, 2011.

[173] A. Sharma and K. K. Paliwal. Fast principal component analysis using fixed-point algorithm. *Pattern Recognition Letters*, 28(10):1151–1155, 2007.

[174] L. Shi, S. Yu, W. Lou, and Y. T. Hou. Sybilshield: An agent-aided social network-based sybil defense among multiple communities. In *Proceedings of the 32nd Conference on Information Communications (INFOCOM)*, 2013.

[175] R. R. Sillito and R. B. Fisher. Semi-supervised learning for anomalous trajectory detection. In *Proceedings of the British Machine Vision Conference 2008 (BMVC)*, 2008.

[176] T. Stein, E. Chen, and K. Mangla. Facebook immune system. In *Proceedings of the 4th Workshop on Social Network Systems (SNS)*, 2011.

[177] A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining partitionings. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, 2002.

[178] G. Stringhini, M. Egele, C. Kruegel, and G. Vigna. Poultry markets: on the underground economy of twitter followers. In *Proceedings of the 2012 ACM workshop on Workshop on Online Social Networks*, 2012.

[179] G. Stringhini, C. Kruegel, and G. Vigna. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC'10)*, 2010.

[180] G. Stringhini, G. Wang, M. Egele, C. Kruegel, G. Vigna, H. Zheng, and B. Y. Zhao. Follow the green: growth and dynamics in twitter follower markets. In *Proceedings of the 2013 conference on Internet Measurement Conference (IMC)*, 2013.

[181] E. Tan, L. Guo, S. Chen, X. Zhang, and Y. Zhao. Unik: unsupervised social network spam detection. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management (CIKM)*, 2013.

[182] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time url spam filtering service. In *Proceedings of the 2011 32nd IEEE Symposium on Security and Privacy (IEEE S&P)*, 2011.

[183] K. Thomas, D. McCoy, C. Grier, A. Kolcz, and V. Paxson. Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse. In *Proceedings of the 22nd USENIX Security Symposium (USENIX Security)*, 2013.

[184] N. Tran, J. Li, L. Subramanian, and S. S. Chow. Optimal Sybil-resilient node admission control. In *Proceedings of the 30th Conference on Information Communications (INFOCOM)*, 2011.

[185] N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-resilient online content voting. In *Proceedings of the 6th Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[186] Trulytweeting service. `http://trulytweeting.app-ns.mpi-sws.org/`.

[187] P. Tsuchiya. The landmark hierarchy: A new hierarchy for routing in very large networks. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 1988.

[188] New Tweets per second record, and how! `https://blog.twitter.com/2013/new-tweets-per-second-record-and-how`.

[189] Twitter usage statistics. `https://about.twitter.com/company`.

[190] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[191] B. Viswanath, M. A. Bashir, M. Crovella, S. Guha, K. P. Gummadi, B. Krishnamurthy, and A. Mislove. Towards detecting anomalous user behavior in online social networks. In *Proceedings of the 23rd USENIX Security Symposium (Usenix Security)*, 2014.

[192] B. Viswanath, M. A. Bashir, M. B. Zafar, S. Bouget, S. Guha, K. P. Gummadi, A. Kate, and A. Mislove. Strength in numbers: Robust tamper detection in crowd

computations. In *Proceedings of the 3rd ACM Conference on Online Social Networks (COSN)*, 2015.

[193] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in Facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN)*, 2009.

[194] B. Viswanath, M. Mondal, A. Clement, P. Druschel, K. P. Gummadi, A. Mislove, and A. Post. Exploring the design space of social network-based Sybil defense. In *Proceedings of the 4th International Conference on Communication Systems and Network (COMSNETS)*, 2012.

[195] B. Viswanath, M. Mondal, K. P. Gummadi, A. Mislove, and A. Post. Canal: Scaling social network-based Sybil tolerance schemes. In *Proceedings of the 7th European Conference on Computer Systems (EuroSys)*, Bern, Switzerland, April 2012.

[196] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove. An analysis of social network-based Sybil defenses. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2010.

[197] A. H. Wang. Don't follow me: Spam detection in twitter. In *Proceedings of the 2010 International Conference on Security and Cryptography (SECRYPT)*, 2010.

[198] G. Wang, T. Konolige, C. Wilson, X. Wang, H. Zheng, and B. Y. Zhao. You are how you click: Clickstream analysis for sybil detection. In *Proceedings of the 22nd USENIX Security Symposium (Usenix Security)*, 2013.

[199] G. Wang, T. Wang, H. Zheng, and B. Y. Zhao. Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers. In *Proceedings of the 23rd USENIX Security Symposium (Usenix Security)*, 2014.

[200] G. Wang, C. Wilson, X. Zhao, Y. Zhu, M. Mohanlal, H. Zheng, and B. Y. Zhao. Serf and turf: crowdturfing for fun and profit. In *Proceedings of the 21st International conference on World Wide Web (WWW)*, 2012.

[201] W. Wei, F. Xu, C. C. Tan, and Q. Li. Sybildefender: Defend against sybil attacks in large social networks. In *Proceedings of the 31st Conference on Information Communications (INFOCOM)*, 2012.

[202] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *Proceedings of the 4th ACM European Conference on Computer systems (EuroSys)*, 2009.

[203] G. Wu, D. Greene, B. Smyth, and P. Cunningham. Distortion as a validation criterion in the identification of suspicious reviews. In *Proceedings of the First Workshop on Social Media Analytics*, 2010.

[204] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming Botnets: Signatures and Characteristics. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)* , 2008.

[205] J. Xue, Z. Yang, X. Yang, X. Wang, L. Chen, and Y. Dai. Votetrust: Leveraging friend invitation graph to defend against social network sybils. In *Proceedings of the 32nd Conference on Information Communications (INFOCOM)*, 2013.

[206] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai. Uncovering social network sybils in the wild. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(1):2, 2014.

[207] Yelp usage statistics. `http://www.yelp.com/factsheet`.

[208] H. Yu. Sybil defenses via social networks: A tutorial and survey. *SIGACT News*, 42(3), 2011.

[209] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A near-optimal social network defense against Sybil attacks. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy (IEEE S&P)*, 2008.

[210] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: Defending against Sybil attacks via social networks. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2006.

[211] H. Yu, C. Shi, M. Kaminsky, P. B. Gibbons, and F. Xiao. DSybil: Optimal sybil-resistance for recommendation systems. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy (IEEE S&P)*, 2009.