UNIVERSITY OF CALIFORNIA

Santa Barbara

Stepping Up the Cybersecurity Game: Protecting Online Services from Malicious Activity

A Dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science

by

Gianluca Stringhini

Committee in charge:

Professor Christopher Kruegel, Chair

Professor Giovanni Vigna

Professor Ben Y. Zhao

June 2014

The dissertation of Gianluca Stringhini is approved.

Professor Ben Y. Zhao

Professor Giovanni Vigna

Professor Christopher Kruegel, Chair

June 2014

Stepping Up the Cybersecurity Game:

Protecting Online Services from Malicious Activity

by

Gianluca Stringhini

ACKNOWLEDGMENTS

Many people supported and helped me during this long journey called PhD.

I would like to thank my adviser Christopher Kruegel and Giovanni Vigna, for mentoring me throughout my PhD. Their advice has been extremely important to teach me what research is all about, and to overcome the many difficulties that I encountered during my projects. I would also like to thank Ben Zhao and Dick Kemmerer for always being there for me and providing me with great advice both in research and during my job hunt. I am also thankful to Janet Kayfetz, who spent many hours helping me improve my writing and presenting skills.

This work would have not been possible without my coauthors, both the ones at UCSB and the ones far away. I would like to thank all of them for working together with me, and for the fruitful conversations in real life or over the Internet. I would also like to thank the folks at IBM Research, Symantec, and Yahoo! for their support, and for showing me what research in the industry is like.

I am thankful to all the people that have been part of the Seclab during these years. It is the best lab that one could wish for, and a great environment to work in. I will always remember the long deadline nights, the CTF competitions, and all the conversations that we had during my stay at UCSB. I have also been very fortunate to have great friends (a.k.a. the Seclab extended family) who made these five years memorable. You guys rock, thanks for all the adventures, the partying, and the good times.

Finally, I would like to thank my family, for always supporting and encouraging me from far away.

VITA OF GIANLUCA STRINGHINI

June 2014

EDUCATION

University of California, Santa Barbara
Ph.D., Computer Science
Santa Barbara, CA USA
September 2009 – June 2014

University of California, Santa Barbara
M.S., Computer Science
Santa Barbara, CA USA
September 2009 – January 2014

Universitá degli Studi di Genova
Laurea Specialistica (M.S. Equivalent) in Computer Engineering
Genova, Italy
September 2006 – March 2009

Universitá degli Studi di Genova
Laurea Triennale (B.S. Equivalent) in Computer Engineering
Genova, Italy
September 2003 – September 2006

ACADEMIC EXPERIENCE

University of California, Santa Barbara
Research Assistant, Computer Security Lab
Santa Barbara, CA USA
September 2009 – June 2014
Advised by Professor Christopher Kruegel

PROFESSIONAL EXPERIENCE

Yahoo! Inc.
Research Contractor
Sunnyvale, CA USA
November 2013 – May 2014

- Developed novel techniques to detect large-scale malicious activity on online services

Symantec Corp.
Research Intern
San Francisco, CA USA
February 2013 – May 2013

- Developed novel techniques to detect targeted email attacks

IBM T.J. Watson Research Center
Visiting Researcher
Yorktown Heights, NY USA
September 2012 – December 2012

- Developed techniques to detect targeted attacks against corporate networks

PUBLICATIONS

Detecting Spammers on Social Networks
Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna
In Proceedings of the 2010 Annual Computer Security Application Conference (ACSAC), Austin, TX.

The Underground Economy of Spam: A Botmaster's Perspective of Coordinating Large-Scale Spam Campaigns
Brett Stone-Gross, Thorsten Holz, Gianluca Stringhini, and Giovanni Vigna
In Proceedings of the 2011 USENIX Workshop on Large-Scale Exploits and Emerging Threats (LEET), Boston, MA.

BotMagnifier: Locating Spambots on the Internet
Gianluca Stringhini, Thorsten Holz, Brett Stone-Gross, Christopher Kruegel, and Giovanni Vigna
In Proceedings of the 2011 USENIX Security Symposium, San Francisco, CA, 2011.

Hit 'em Where it Hurts: A Live Security Exercise on Cyber Situational Awareness
Adam Doupe, Manuel Egele, Benjamin Caillat, Gianluca Stringhini, Gorkem Yakin, Ali Zand, Ludovico Cavedon, and Giovanni Vigna
In Proceedings of the 2011 Annual Computer Security Applications Conference (ACSAC), Orlando, FL.

B@bel: Leveraging Email Delivery for Spam Mitigation
Gianluca Stringhini, Manuel Egele, Apostolis Zarras, Thorsen Holz, Christopher Kruegel, and Giovanni Vigna
In Proceedings of the 2012 USENIX Security Symposium, Bellevue, WA, 2012.

Poultry Markets: On the Underground Economy of Twitter Followers
Gianluca Stringhini, Manuel Egele, Christopher Kruegel, and Giovanni Vigna
In Proceedings of the 2012 SIGCOMM Workshop on Online Social Networks (WOSN), Helsinki, Finland, 2012.

PUBLICATIONS (CONTINUED)

COMPA: Detecting Compromised Accounts on Social Networks
Manuel Egele, Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna
In Proceedings of the 2013 ISOC Network and Distributed Systems Symposium (NDSS), San Diego, CA.

Two Years of Short URLs Internet Measurement: Security Threats and Countermeasures
Federico Maggi, Alessandro Frossi, Gianluca Stringhini, Brett Stone-Gross, Christopher Kruegel, Giovanni Vigna, and Stefano Zanero
In Proceedings of the 2013 International World Wide Web Conference (WWW), Rio De Janeiro, Brazil.

Follow the Green: Growth and Dynamics in Twitter Followers Markets
Gianluca Stringhini, Gang Wang, Manuel Egele, Christopher Kruegel, Giovanni Vigna, Ben Y. Zhao, and Haitao Zheng
In Proceedings of the 2013 Internet Measurement Conference (IMC), Barcelona, Spain.

Shady Paths: Leveraging Surfing Crowds to Detect Malicious Web Pages
Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna
In Proceedings of the 2013 ACM Conference on Computer and Communication Security (CCS), Berlin, Germany.

Stranger Danger: Exploring the Ecosystem of Ad-based URL Shortening Services
Nick Nikiforakis, Federico Maggi, Gianluca Stringhini, M Zubair Rafique, Wouter Joosen, Christopher Kruegel, Frank Piessens, Giovanni Vigna, and Stefano Zanero
In Proceedings of the 2014 International World Wide Web Conference (WWW), Seoul, Korea.

The Harvester, the Botmaster, and the Spammer: On the Relations Between the Different Actors in the Spam Landscape
Gianluca Stringhini, Oliver Hohlfeld, Christopher Kruegel, and Giovanni Vigna
In Proceedings of the 2014 ACM Symposium on Information, Computer and Communications Security (ASIACCS), Kyoto, Japan.

PUBLICATIONS (CONTINUED)

The Tricks of the Trade: What Makes Spam Campaigns Successful?

Jane Iedemska, Gianluca Stringhini, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna

In Proceedings of the 2014 International Workshop on Cybercrime (IWCC), San Jose, CA.

ABSTRACT

Stepping Up the Cybersecurity Game:
Protecting Online Services from Malicious Activity

by

Gianluca Stringhini

The rise in popularity of online services such as social networks, web-based emails, and blogs has made them a popular platform for attackers. Cybercriminals leverage such services to spread spam, malware, and steal personal information from their victims. In a typical cybercriminal operation, miscreants first infect their victims' machines with malicious software and have them join a botnet, which is a network of compromised computers. In the second step, the infected machines are often leveraged to connect to legitimate online services and perform malicious activities.

As a consequence, online services receive activity from both legitimate and malicious users. However, while legitimate users use these services for the purposes they were designed for, malicious parties exploit them for their illegal actions, which are often linked to an economic gain. In this thesis, I show that the way in which malicious users and legitimate ones interact with Internet services presents differences. I then develop

mitigation techniques that leverage such differences to detect and block malicious parties that misuse Internet services.

As examples of this research approach, I first study the problem of spamming botnets, which are misused to send hundreds of millions of spam emails to mailservers spread across the globe. I show that botmasters typically split a list of victim email addresses among their bots, and that it is possible to identify bots belonging to the same botnet by enumerating the mailservers that are contacted by IP addresses over time. I developed a system, called BOTMAGNIFIER, which learns the set of mailservers contacted by the bots belonging to a certain botnet, and finds more bots belonging to that same botnet.

I then study the problem of misused accounts on online social networks. I first look at the problem of fake accounts that are set up by cybercriminals to spread malicious content. I study the *modus operandi* of the cybercriminals controlling such accounts, and I present a system to automatically flag a social network accounts as fake. I then look at the problem of legitimate accounts getting compromised by miscreants, and I present COMPA, a system that learns the typical habits of social network users and considers messages that deviate from the learned behavior as possible compromises.

As a last example, I present EVILCOHORT, a system that detects communities of online accounts that are accessed by the same botnet. EVILCOHORT works by clustering together accounts that are accessed by a common set of IP addresses, and can work on any online service that requires the use of accounts (social networks, web-based emails, blogs, etc.).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Cybercrime is a serious threat for Internet users. Miscreants infect their victim computers and control them to perform malicious activities, such as sending email spam [125], stealing their victims' personal information [123], performing denial of service attacks [59], or mining digital currencies [71]. Cybercrime operations are a successful business, generating important revenues for attackers [77, 78].

To perform malicious activity on online services, such as online social networks, web-based email services, and blog platforms, cybercriminals need access to three elements: *infected machines*, a *command and control infrastructure*, and *online accounts*. Cybercriminals can either infect computers themselves to take control of them [108, 125], or can purchase already-compromised machines on the black market [38]. After taking control of a number of compromised machines, the attacker instructs them to connect to a server under his control — the infected machine becomes a *bot* and joins a botnet,

which is a network of compromised computers controlled by the same cybercriminal through a *command and control* (C&C) server [51]. A cybercriminal controlling a botnet is typically referred to as a *botmaster*. The C&C server is then used by the cybercriminal to issue commands to his bots. A common action performed by botmasters is to instruct their bots to connect to accounts under their control on online services, and perform malicious activity such as spreading spam, malware, or crawling sensitive information. These accounts can be fake ones, created specifically for malicious purposes [138] or legitimate accounts that have been compromised and are now under the cybercriminal's control [63].

Leveraging existing services to spread malicious content provides three advantages to the cybercriminal. First, it is easy to reach many victims, since popular online services have many millions of users that are well connected. Consider traditional email spam operations, where miscreants have to harvest a large number of victim email addresses (on the web or from infected hosts) before they can start to send spam [128]. On social networks, cybercriminals can easily find and contact their victims or leverage the existing friends of legitimate accounts that they compromised [63]. In some cases, such as blog and forum spam, cybercriminals do not even have to collect a list of victims, because their malicious content will be shown to anybody who is visiting the web page on which the spam comment is posted [101, 139]. A second advantage of using online services to spread malicious content is that while users have become aware of the threats associated with email, they are not as familiar with scams and spam that spreads through more recent channels (such as social networks) [75, 35]. The third advantage is that while online services have good defenses against threats coming from the outside, they have a much harder time in detecting misuse that originates from accounts within the

service itself [136].

In this dissertation I present my contribution to fighting cybercriminal operations. My work follows a data-driven approach to detect and block malicious activity on online services. In my research, I observe the way in which infected machines interact with legitimate online services, compared to regular users. Such interactions will show differences, because malicious and legitimate users use online services for very different purposes: while miscreants want to make a profit from interacting with legitimate services, and therefore aim at performing as much malicious activity as they can, regular users stick to a fair use of the online service, using it in the way the service was designed.

After identifying differences in the interaction behavior that clearly distinguish malicious and legitimate activity, I design systems that leverage them to detect and block this malicious activity. As examples of this research approach, I present four projects that fight different types of malicious activity on online services:

- BOTMAGNIFIER, a system that learns the email-sending behavior of the spamming bots belonging to a botnet, and is able to find more bots belonging to the same botnet in the wild.
- SPAMDETECTOR, a system that learns the typical modus operandi of miscreants that leverage fake accounts on social networks, and is able to detect such accounts.
- COMPA, a system that is able to detect whether an online social network account has been compromised and is now misused by a cybercriminal.

- EVILCOHORT, a system that studies the relations between infected computers and online accounts, and is able to detect communities of accounts on an online service that are accessed by the same botnet.

In the following, I first describe the elements and the characteristics of a typical cybercriminal operation. Then, I provide an overview of this dissertation work in fighting them. Finally, I present the contributions of this thesis.

1.1 Anatomy of a Cybercriminal Operation

As I mentioned, a cybercriminal operation is typically composed of three parts: a set of *infected machines*, a *command and control infrastructure* that allows the cybercriminal to give orders to his bots, and a set of *online accounts*, which are used by the bots to connect to online services and perform malicious activity, such as spreading spam. Although online accounts are not required in all cases, they are usually needed if the infected machine is required to access restricted information on the service, or post content. In the following, I describe the three elements needed by cybercriminals to run their operations, and illustrate the options that they have for each of these elements, as well as the challenges and drawbacks associated to them.

1.1.1 Infected Machines

Ideally, a cybercriminal would want access to as many infected machines as possible. The first infections that happened on the Internet belonged to the class of *worms* [80]. In this type of threat, machines would get infected by having a vulnerability in a software component compromised. After the machine becomes infected, it would scan the network for other vulnerable machines, and try to exploit them. Worm infections are characterized by the fact that they are spreading autonomously, and often time this spread got out of control. In 1988, the first worm ever developed resulted in generating so much traffic that the Internet backbone had to be restarted [80]. This particular threat, known as the "Morris worm," did not aim to turn a profit or generate any malicious activity other than spreading, and was designed to show the feasibility of an Internet-wide infection. More recent threats, which had large denial of service attacks against webservers as a goal, spread out of control too and were detected before they could reach their goal [156].

Not only worm activity tends to be very obvious and easy to detect, but recent research showed that a botnet can effectively manage only a certain number of bots, before the command and control channel gets saturated [73] — a botnet composed of two many bots results in having a part of them remaining idle. Due to these practical issues – the lack of stealthiness and the uselessness of having more infected machines than needed – cybercriminals stopped using the worm-like spread to collect populations of infected computers; the last popular worm was `Conficker`, which dates back to 2008 [118].

The methods of collecting infected machines that were developed more recently do not

involve self-spreading infections. Instead, malware is installed by having users click on a malicious attachment in spam emails [125] or by having the user visit a malicious web page, which would try to exploit vulnerability in the user's browser and install a malicious program without the user noticing [108]. A third type of distribution involves having the user visit a scam web page that convinces her to install a malware program, disguised as a useful tool (such as an antivirus program) [122].

Nowadays, an entire economy emerged that is specialized in providing infected computers to cybercriminals. Miscreants set up their own botnet, which is able to install a malware of of their customers' choice on demand [38]. The infections used by such botnets are usually very resilient and difficult to eradicate — for example the malicious program could reside in the master boot record (MBR) of a hard drive and patch the operating system kernel at each reboot to remain undetected [124]. The sophistication of this market has evolved to the point that customers can specify the type of infected machines that they want to buy, in terms of operating system, configuration, and geographic location. Bots located in developed countries (e.g., the United States or the United Kingdom) are more expensive than bots located in other countries [38]. This might be linked to the fact that such machines are more valuable for particular cybercriminal schemes, such as stealing banking information. However, recent research showed that for traditional cybercriminal activity, such as sending spam, there is no advantage for a miscreant to buy bots in expensive countries [73].

Computer infections are a real problem in nowadays Internet. A recent report by Symantec shows that in 2013 the company detected 2.3 billion infected machines [135]. In Chapter 2.2 we provide an overview of the techniques that have been proposed to

detect and clean up infected computers, while in Chapter 3 we present a novel system to detect infected computers that are used to spread email spam.

1.1.2 Command and Control Infrastructure

After a cybercriminal has collected a number of infected computers, he instructs them to connect to a command and control (C&C) infrastructure under his control. This infrastructure can be a single server [125], or a multitude of communicating servers [44].

Botnets that take advantage of a single server can either leverage a well-known protocol, such as IRC [25], or a proprietary, encrypted protocol [125]. Using a proprietary protocol makes it more difficult for researchers to intercept the commands issued by the botmaster, but the fact that the operation uses only one C&C server constitutes a single point of failure, and security researchers can take down the entire botnet by seizing this server. To be more resilient, other botnets use multiple C&C servers. In particular, a multi-tier model is commonly used: a unique C&C server is hidden through multiple layers of servers whose only purpose is relaying connections between the bots and the upper levels of the C&C infrastructure [44]. The purpose of the relay servers is to hide the actual *mothership*, which is the server from which the botmaster issues commands, and make takedowns difficult: as long as a majority of relays and the mothership are still available, the botnet remains operational.

Some botnets obfuscate their infrastructure even more, by using a peer-to-peer scheme. In this model, the bots themselves act as relays between the infected machines and the upper C&C infrastructure [102]. Although these schemes make it more difficult

for researchers to identify the C&C servers, this type of infrastructure has the problem that a good fraction of the command and control traffic passes through the infected machines. Researchers can then pretend to be bots and *infiltrate* the botnet, obtaining important insights on the entire cybercriminal operation [44, 102, 121].

Another technique that cybercriminals can use to obfuscate their operations is modifying the IP address that corresponds to a certain domain often, with a technique called *fast flux* [69]. In this model, a set of relay servers is associated to the same DNS domain in quick sequence, one at a time. The infected machines contacting the C&C infrastructure do that by issuing DNS queries to the domain, and therefore the botnet can work regardless the relay that is active at the moment. As long as one of the relays associated to the domain is active, the operation keeps being alive, and the bots are able to receive their orders.

A more recent technique that makes the detection of C&C infrastructures even more difficult is dynamically changing the DNS domain to access the infrastructure over time. By leveraging a time-dependent algorithm, known as *domain generation algorithm* (DGA), the bots can dynamically generate the domain that is active at a certain point in time, and connect to the C&C infrastructure [28].

In Chapter 2.2 we discuss the different efforts conducted by the research community to disrupt C&C infrastructures. As we mentioned previously, in this dissertation we focus on developing mitigation systems based on the way in which infected machines interact with legitimate, online services. Since C&C infrastructures are entirely composed of servers controlled by cybercriminals, studying them goes beyond the scope of this thesis.

1.1.3 Malicious Online Service Accounts

To interact with many online services, such as online social networks and web-based email services, cybercriminals need access to accounts on such services. These accounts can be used to spread malicious content on the service, such as spam and links pointing to malware [63], or to connect to the service and crawl sensitive information, such as user profile data [74]. The accounts that cybercriminals can take advantage of to spread malicious content can be *fake accounts*, which have been specifically created for that purpose, or *compromised accounts*, which belong to legitimate users but have been taken over by the cybercriminal.

To obtain fake accounts on an online service, miscreants can either create them themselves, pay workers on crowdsourcing platforms to create a number of them [145], or purchase a bulk of already-created ones on the black market [138]. These accounts can then be used to spread malicious content on the online service. A problem with fake accounts is that many online services, such as social networks, require users to build a network of contacts to share content. A fake account is problematic in this regard, since real users are less likely to accept contact requests from people they do not know [35]. Also, trying to establish a large number of social connections in a short period of time is suspicious, and can lead to the fake account being detected and deleted by the online service [150]. In the case of online social networks, an alternative that cybercriminals can attempt to quickly acquire contacts is purchasing friends or followers online: there are many services that promise to provide a large number of contacts to their customers [127, 132]. Often times, these contacts come from legitimate accounts that have been compromised. In Chapter 5 we will present SPAMDETECTOR, our solution

to detect fake accounts misused by cybercriminals on online social networks.

As an alternative to fake accounts, cybercriminals can leverage legitimate online accounts that have been compromised. A compromised account offers multiple advantages for cybercriminals compared to a fake one. First of all, a legitimate account already has an established network of trust, and therefore the cybercriminal does not need to create one. In addition, the friends of the compromised account's legitimate owner are likely to trust her, and therefore it is more likely for them to click on the links that the account posts [75]. As an additional advantage, from the social network perspective dealing with compromised accounts is more difficult than dealing with fake accounts: while a fake account can just be suspended with no consequences, a compromised account needs special handling, such as resetting the account password and informing the account's legitimate owner about what happened.

Compromised accounts can be stolen by having their owners give away credentials through a phishing scam [22], by luring users into authorizing a rogue application to use the accounts through social engineering [132], or by leveraging a vulnerability in the web application that powers the online service (for example, an XSS vulnerability) [143]; in this case, the user automatically posts some content of the attacker's choice by displaying the malicious content in her web browser. Due to the automatic exploitation of such vulnerabilities, the messages generated by an XSS vulnerability tend to spread in an uncontrollable fashion (similar to the typical worm behavior that we discussed in Chapter 1.1.1) [41]. In chapter 6 we present COMPA, the first system to detect legitimate online social network accounts that have been compromised. This work represents the first step in ensuring that messages that spread on online social

networks are really authored by who they claim to come from.

1.2 Dissertation Overview

This dissertation collects the contributions of four papers: "BOTMAGNIFIER: Locating Spambots on the Internet" [129] (USENIX Security 2011), "Detecting Spammers on Social Networks" [130] (ACSAC 2010), "COMPA: Detecting Compromised Accounts on Social Networks" [57] (NDSS 2013), and "EVILCOHORT: Detecting Communities of Malicious Accounts on Online Services" (under submission). These systems detect and block various types of malicious activity aimed at online services, at three different levels: they either look at infected hosts, at accounts that are misused by cybercriminals, or at the relation between infected hosts and malicious accounts. In the following, I illustrate a summary of the aforementioned systems.

1.2.1 Detecting Misbehaving Hosts

One way of disrupting cybercriminal operations is developing techniques to detect and block the infected machines that form botnets. A common task that cybercriminals use botnets for is sending spam emails to their victims. Typically, a botmaster sets up a botnet by infecting victim computers, and rents it out to spammer groups to carry out spam campaigns, in which similar email messages are sent to a large group of Internet users in a short amount of time. Tracking the bot-infected hosts that participate in spam campaigns, and attributing these hosts to spam botnets that are active on the Internet, are

challenging but important tasks. In particular, this information can improve blacklist-based spam defenses and guide botnet mitigation efforts.

In Chapter 3 I present a novel technique to support the identification and tracking of bots that send spam. Our technique takes as input an initial set of IP addresses that are known to be associated with spam bots, and learns their spamming behavior. This initial set is then "magnified" by analyzing large-scale email delivery logs to identify other hosts on the Internet whose behavior is similar to the behavior previously modeled. We implemented our technique in a tool, called BOTMAGNIFIER, and applied it to several data streams related to the delivery of email traffic. Our results show that it is possible to identify and track a substantial number of spam bots by using our magnification technique — Over a period of four months, we were able to grow the set of known bots belonging to well-known botnets by 925,978 IP addresses, which accounts for 45.6% of the total. We also tracked the evolution and activity of such spamming botnets over time. Moreover, we show that our results can help to improve state-of-the-art spam blacklists.

1.2.2 Detecting Misbehaving Accounts

An orthogonal way of disrupting threats against online services is detecting and blocking the accounts that are misused by cybercriminals. In this dissertation I present two systems that detect malicious accounts on online social networks: the first one, SPAMDETECTOR, detects accounts that are specifically created to spread spam on the network. The second one, COMPA, detects legitimate accounts that have been taken

over by a cybercriminal and that are used to spread malicious content.

Detecting fake social network accounts. Social networking has become a popular way for users to meet and interact online. Users spend a significant amount of time on popular social network platforms, storing and sharing a wealth of personal information. This information, as well as the possibility of contacting thousands of users, also attracts the interest of cybercriminals. In particular, as we mentioned earlier in this chapter, cybercriminals take advantage of fake accounts to disseminate malicious content on social networks.

In Chapter 5, we first analyze to which extent spam has entered social networks. More precisely, we analyze how spammers who target social networking sites operate. To collect the data about spamming activity, we created a large and diverse set of "honey-profiles" on three large social networking sites (Facebook, Twitter, and MySpace), and logged the kind of contacts and messages that they received. We then analyze the collected data and model the anomalous behavior of the fake accounts contacted our profiles. We developed a system, called SPAMDETECTOR, which detects fake accounts that are misused by cybercriminals to spread malicious content on social networks based on the typical behavior showed by these accounts. SPAMDETECTOR was one of the first systems that were presented to detect fake accounts on social networks.

We then aggregate the messages sent by the detected accounts in large spam campaigns. Our results show that it is possible to automatically identify the accounts used by spammers, and our analysis was used for take-down efforts in a real-world social network. More precisely, during this study, we collaborated with Twitter and correctly detected and deleted 15,857 spam profiles.

I present SPAMDETECTOR in detail, alongside with our results, in Chapter 5.

Detecting compromised social network accounts. As the results that we obtained with SPAMDETECTOR show, fake accounts typically exhibit highly anomalous behavior, and hence, are relatively easy to detect. As a response, attackers have started to compromise and abuse legitimate accounts. Compromising legitimate accounts is very effective, as attackers can leverage the trust relationships that the account owners have established in the past. Moreover, compromised accounts are more difficult to clean up because a social network provider cannot simply delete the corresponding profiles.

In Chapter 6 of this dissertation, we present the first approach to detect compromised user accounts in social networks, and we apply it to two popular social networking sites, Twitter and Facebook. Our approach uses a composition of statistical modeling and anomaly detection to identify accounts that experience a sudden change in behavior. Since behavior changes can also be due to benign reasons (e.g., a user could switch her preferred client application or post updates at an unusual time), it is necessary to derive a way to distinguish between malicious and legitimate changes. To this end, we look for groups of accounts that all experience similar changes within a short period of time, assuming that these changes are the result of a large-scale malicious campaign that is unfolding. We developed a tool, called COMPA, that implements our approach, and we ran it on a large-scale dataset of more than 1.4 billion publicly-available Twitter messages, as well as on a dataset of 106 million Facebook messages. COMPA was able to identify compromised accounts on both social networks with high precision. In particular, over a period of three months, COMPA was able to detect more than 300,000 compromised accounts on Twitter. We worked closely with the Twitter security team

to make sure that these accounts were cleaned up. In addition, we show that COMPA would have helped in detecting two high-profile compromises against news agency accounts that happened on Twitter in 2011 and in 2013.

I present COMPA in detail in Chapter 6.

1.2.3 Studying the Relations Between Bots and Malicious Accounts

So far, I have presented techniques that either focus on detecting malicious hosts or malicious accounts on online services. In Chapter 7 I present the first system that detects malicious accounts on online services by studying the relations between the accounts controlled by a cybercriminal and the botnet (i.e., the set of IP addresses) that accesses them.

We first study the accounts that were used to send spam on a large web-based email service, and show that accounts that are accessed by botnets are particularly dangerous because of the amount of malicious activity that they can carry out and their ability to survive for long periods of time. Since botnets are composed by a finite number of infected computers, we observe that cybercriminals tend to have their bots connect to multiple online accounts to perform malicious activity. We present EVILCOHORT, a system that detects online accounts that are accessed by the same botnet. We evaluated EVILCOHORT on multiple online services: first, we ran it on a dataset collected on a large web-based email service, and were able to detect 1.2 million malicious accounts over a period of five months. We also ran the system on a dataset collected on multiple social networks, and were able to detect 111,647 malicious accounts over a period of 8

days.

Our results show that EVILCOHORT can reliably detect malicious accounts that are accessed by botnets. In addition, EVILCOHORT opens exciting scenarios in the area of malicious account detection: unlike previous systems that can detect malicious accounts only once they send malicious content or interact with the online service (for example by sending a friend request to a victim), our system only looks at the set of IP addresses that access a set of accounts. Therefore, EVILCOHORT can detect accounts that, although controlled by cybercriminals, do not perform any "clearly" malicious activity on the online service. An example are botnets that use the online service as command and control channel, and retrieve commands from the network without posting anything.

In Chapter 7 I provide a detailed description of EVILCOHORT, and I show detailed statistics on the communities of malicious accounts that we detected.

1.3 Contributions

In summary, this dissertation makes the following contributions:

- I present a system to increase the knowledge on the set of bots belonging to a botnet. This system can be used to track the population of large botnets, and record important events in the botnet's lifetime (such as a takedown, or a botnet becoming active again after a period of inactivity). In addition, our system is a useful aid to improve DNS blacklists, which are known to have coverage problems [109].

- I present one of the first systems to detect and block fake accounts that are misused by cybercriminals on online social networks. The system reaches very high precision, and is a helpful tool for social network operators to keep their networks safe.

- I present the first system able to detect that an online social network account has been compromised. In addition to detecting large-scale compromises, in which thousands of accounts are involved and misused by botnets, this system would have been helpful in preventing two high-profile social network accounts belonging to news agencies from being compromised and post unwanted content.

- I present an analysis of the relation between the set of IP addresses accessing an online account and its maliciousness. I show that accounts that are accessed by botnets are particularly dangerous for the online service, and present a system to detect and block such accounts.

Chapter 2

Related Work

Cybercriminals and security researchers are involved in an arms race. Every time a new mitigation technique is deployed, a more advanced attack or botnet scheme is developed. In this chapter, I first analyze the evolution that botnets followed over the years. Then, I describe the detection and mitigation systems that have been developed by the security community to fight the threat of cybercriminal operations.

2.1 The Evolution of Botnets

Since they first became a threat, in the mid-2000s, botnets evolved considerably, to keep up with the security research being conducted to disrupt them and make sure their operation could continue. This evolution happened in two aspects: the structure of the botnets and the way they propagate their infections to acquire new victims.

2.1.1 The Evolution of Botnet Structures

IRC botnets. The first botnets borrowed characteristics from Internet Relay Chat (IRC) bots. After being infected, a bot would connect to an IRC server, join a specific channel, and wait for orders [25, 54, 55]. These botnets did not use a lot of sophistication to hide their actions. They usually used a password to protect the IRC channel where the botmaster would give the orders, but both the server name and the password would be in clear in the malicious binary. Therefore, researchers could retrieve this information and join the channel to learn important information about the botnet. As an alternative, researchers could sinkhole the DNS traffic asking for the malicious domains, so that the infected machines would connect to them instead of going to the Command and Control (C&C) server [25]. Another weakness of this model is that the C&C traffic used the IRC protocol, which is easy to detect and monitor.

Proprietary botnets. To avoid using known protocols and make their activity less evident, botmasters started using proprietary, encrypted protocols for their C&C traffic [125]. This makes botnet infiltration more difficult, but researchers can still reverse engineer the protocol, create software that implements it, and join the botnet. In addition, since the infrastructure still uses a single domain, sinkholing remains possible.

Multiple tier botnets. To make their infrastructure more resilient to attacks, botmasters started developing multiple-tier botnets [123]. In this architecture, instead of contacting the C&C server directly, the bots contact one of many proxies, which then forward their request to the C&C server. By doing this cybercriminals make sure that, even if researchers took control of a limited number of proxies, the botnet would still be

operational.

In addition, botmasters developed a new technique that gives them even more reliability on their C&C infrastructure: *fast flux* [69, 103]. This technique is similar to the ones involved in Round Robin DNS and the ones used by Content Delivery Networks, and its goal is to assign a fast-changing number of IP addresses to the domains used by the botnet C&C infrastructure. By doing this, the botmaster ensures that even if a very small number of proxies would survive a takedown, the botnet would still be operational.

Even if the C&C infrastructure in this scheme uses multiple domains, it is still possible for researchers to sinkhole or blacklist them.

Domain Generation Algorithms. To mitigate the disadvantages of having a limited number of domains for the C&C infrastructure, cybercriminals developed algorithms that allow both the bots and the C&C to generate domains on the fly. These algorithms, called Domain Generation Algorithms (DGAs), are typically time-sensitive, and, at any point in time, tell bots at which domain they can find the C&C server (or one of the proxies associated to it) [123]. The botmaster has to register the domains that will be used in the future, and make sure his infrastructure will respond to the bots at the right time.

Of course, researchers could reverse engineer the domain generation algorithm, and register the domains before the botmaster does. A countermeasure to this is making the DGA non-deterministic, for example by using information taken from social network trending topics. This way it is impossible to predict the DGA domains long time before

they become active.

Peer-to-peer botnets. Another evolution is to make the botnet peer-to-peer [85, 121]. In this architecture, another level of relayers is deployed between the bots and the proxies. Typically, those bots that do not have a public IP address (i.e., are behind a NAT) act as regular bots, while those that have a public IP address act as relayer bots. Regular bots find the nearest relayer by implementing some sort of Overnet protocol, which is typical of peer-to-peer networks (e.g., Kademlia [96]). A problem of this approach is that the botmaster gives up the control on a critical part of his infrastructure (i.e., the relayers) to infected machines. Researchers could reverse engineer the C&C protocol and infiltrate the botnet by pretending to be a relayer. This would allow them to collect a wealth of information about the botnet, for example enumerate the bots, or collect the spam templates that are delivered to the infected machines.

2.1.2 The Evolution of the Botnet Infection Model

Not only the botnet structure changed over time, but also the infection model that botmasters use to gain control of more machines evolved. At the beginning, bots were behaving like Internet worms [65]. This means that an infected machine would scan for more vulnerable hosts in her network, and try to propagate. This approach became less and less used over time, with Conficker being the last large botnet using it in 2008 [106]. The next step has been to use bots that did not propagate on their own anymore. In this phase, the main channels of infection were two:

- Sending malicious binaries through spam emails, and luring victims into clicking

on them [125].

- Setting up malicious web pages that tried to exploit vulnerabilities in the victim's browser and to download and install a malicious binary (a so-called *drive-by download attack* [108]).

Nowadays, the trend botmasters follow to deploy their bots is to use third party services. These services typically use pre-existing botnets to download additional components (i.e., bots) on the infected machines for a fee [38].

2.2 The Evolution of Mitigation Techniques

To disrupt a botnet that misuses a legitimate online service, researchers can intervene at different levels: mitigation systems can focus on detecting infected machines, on detecting malicious C&C infrastructures, on detecting malicious content (e.g., email spam), or on detecting malicious online service accounts. In the following, I analyze the efforts performed by the research community in each of these four fields.

2.2.1 Detecting Infections

One way of fighting large-scale cybercriminal operations is detecting infected computers before they can cause any harm. In this section I discuss the research that has been conducted to achieve this goal.

Host based detection. A vantage point that researchers can leverage is the victim

host. By looking at the binaries that get installed, one can try to infer whether they are malicious or not. Traditional anti-viruses build signatures (e.g., regular expressions) from known malware, and look for the presence of those signatures in the binaries the user downloads [24]. This technique is not very robust, and previous work showed how the detection can be fooled by simple obfuscations such as inserting NOP instructions and performing code transpositions [46, 47].

Remaining in the field of static analysis, a better approach is to extract semantic information from known malware samples, and look for the same semantics in new samples while performing detection [48, 47]. The issue here is that program equivalence is an undecidable problem. Therefore, even if the proposed systems can cover a number of variations that model the same behavior, it is not guaranteed that this will work for any possible sample. In addition, modern malware comes packed (i.e., encrypted) and decrypts itself at runtime, and this makes static analysis difficult.

Dynamic analysis makes malware analysis easier, because one can look at the program once the decryption has happened. The techniques that have been proposed include modelling the behavior of a program based on the system calls it executes [82], monitoring programs accessing sensitive information while they should not [153], or looking at the buffers allocated by a malware sample to reconstruct the C&C protocol it uses [39]. The problem of dynamic analysis is that running large amounts of malware samples takes time and resources, and cybercriminals can realize that the malware is being run in an analysis environment, and avoid performing any malicious activity [91].

Malicious web pages detection. Another approach used by researchers is looking at malicious web pages that try to compromise the victims browser and automatically

download a piece of malware (in a so-called *drive-by download* attack [108]). These attacks are typically performed by malicious JavaScript scripts. To detect such scripts, various approaches have been used:

- Using machine learning to detect legitimate and malicious web pages. The features that researchers leveraged include how many HTTP redirections the web page uses, or whether the JavaScript code included in the web page is obfuscated. Detection can be performed either offline, by re-visiting the page with an instrumented browser [52], or online, by instrumenting the victim's browser and stopping executing the page once one detects that it is malicious [53, 68, 131].
- Looking at the changes in the victim's system when she visits a malicious web page. The creation of files of the changes of registry keys are indicators of the compromise [93, 108].
- The last possibility is to look at typical attack patterns and flag as malicious any script that shows those patterns [113].

The problem with these techniques is mostly that they rely on a static model to detect malicious scripts, or they train on malicious behavior that might adapt and change over time [72]. Therefore, they could miss newer attacks cybercriminals might come up with.

Network based detection. Another vantage point that can be leveraged for detection is the network traffic generated by infected machines. By observing the malicious traffic generated by such machines it is possible to learn important information about botnets, and develop effective countermeasures.

A direction researchers looked at is detecting successful infections by monitoring network traffic [65]. In this research work, infections are modeled as a set of flows that picture the different steps of the infection. Although interesting, this model cannot be applied anymore today. The reason is that years ago botnet infections followed a well defined, worm-like behavior (i.e., scanning for victims, exploitation, download of an egg, connection to the command and control), which is not widely used anymore.

More recent research proposed to look at the correlation between C&C messages and malicious activity. The idea is that any time a bot will receive a command, it will perform a malicious activity. By looking at this correlation, it is possible to detect bots without any previous knowledge of the botnet [64]. The problem is how to identify C&C traffic. Older approaches looked for commonly-misused, well-known protocols (e.g., IRC) [66]. However, this type of techniques are not applicable anymore, since most botnets moved to proprietary protocols for their C&C traffic. More recent work looks for malicious activity first, and then looks for any interaction with external servers that happened before that activity to find the actual commands [147]. Zand et al. proposed a system to find strings that are typical of C&C commands, and leverage them for detection [155].

DNS based detection. Similar to most legitimate Internet services, botnets use the DNS infrastructure to easily connect the different components of their infrastructure (i.e., the bots and the C&C server). Therefore, by looking at the interaction between bots and DNS servers researchers can learn important information about the botnet, such as which IP addresses are associated to infected machines. This can be done by sinkholing the domains used by a botnet's C&C infrastructure. By doing this, the

infected machines will contact the researchers instead of the botmaster, and it will be possible to enumerate them [55]. Another option is to look in local DNS servers for the presence of cached results associated to malicious domains [25]. If such records are found, that is an indicator of the presence of infected machines in the network.

Ramachandran et al. analyzed queries against a DNS blacklist (DNSBL) to reveal botnet memberships [111]; the intuition behind their approach is that bots might check if their own IP address is blacklisted by a given DNSBL. Such queries can be detected, which discloses information about infected machines.

In Chapter 3 we present BOTMAGNIFIER, a system that can grow the set of known infected machines belonging to a botnet by observing the set of mailservers contacted by such machines over time. As we will see, BOTMAGNIFIER helps in integrating DNS blacklists, significantly improving the coverage offered by these services.

2.2.2 Detecting Command and Control Activity

Other work operated at the Command and Control level. The goal is typically to learn important information about the botnet, or to attempt a takedown.

For older botnets that were using IRC, infiltration was easy. All that researchers had to do was joining the IRC server and a particular channel, and look for the commands being issued by the botmaster [25]. Nowadays, this is not possible anymore, and researchers need to reverse engineer the C&C protocol first. A way of doing that is by active probing [45]. This type of techniques enable botnet infiltration, which means

that researchers can create a piece of software that behaves exactly like a bot, but does not execute any malicious activity [39, 44, 59, 84, 85, 121].

Another possibility, for those botnets that use it, is to reverse engineer the Domain Generation Algorithm (DGA) used by the botnet. By doing this, researchers can register the C&C domains before the botmasters do, and impersonate the C&C server [110, 123]. An orthogonal approach is to take down the C&C server and perform an offline analysis of the server to reconstruct information [102].

In the case of botnets that use fast flux, the domains that use such techniques present very different characteristics from legitimate ones, and can be detected. For example, the IP addresses returned for a DNS query to a fast flux domain will belong to very different networks in various parts of the world, their Time to Live (TTL) will be low, and two subsequent queries will return different results. Previous work focused on building classifiers based on such characteristics, to detect fast flux domains [69, 70, 103].

Another possible way of interacting with the C&C infrastructure is by setting up *honeypots*. Honeypots are virtualized environments where it is possible to run malware, and monitor its activity [76]. Monitoring the activity of the bots can give insights on what the C&C IP addresses and domains are. This information can be used for blacklisting them [126] or for performing botnet takedowns [125]. Honeypots, however, come with some problems. First of all, malware might detect that it is running in a virtualized environment, and do not perform any malicious activity [31]. On the other hand, while running malware, researchers need to make sure that their environment is constrained enough that this malware cannot damage anybody.

One last approach is to detect C&C domains by observing patterns in which domains are queried by infected machines. Proposed systems use local data from Recursive DNS servers [26, 34], or a more comprehensive view by looking at the Top Level Domain (TLD) level [27].

2.2.3 Detecting Malicious Content

Infected computers are commonly used to spread malicious content, such as spam, phishing scams, or links pointing to malicious web pages. In this section I briefly discuss the research that has been performed to detect such malicious content.

Existing work on spam filtering can be broadly classified in two categories: *post-acceptance methods* and *pre-acceptance methods*. Post-acceptance methods receive the full message and then rely on *content analysis* to detect spam emails. There are many approaches that allow one to differentiate between spam and legitimate emails: popular methods include Naive Bayes, Support Vector Machines (SVMs), or similar methods from the field of machine learning [114, 56, 98, 116]. Other approaches for content-based filtering rely on identifying the URLs used in spam messages [148, 14], or in looking at the content of the page pointed by such URLs looking for keywords that are typical of spam [137]. A third method is *DomainKeys Identified Mail* (DKIM), a system that verifies that an email has been sent by a certain domain by using cryptographic signatures [89]. In practice, performing content analysis or computing cryptographic checksums on every incoming email can be expensive and might lead to high load on busy servers [136]. Furthermore, an attacker might attempt to bypass the con-

tent analysis system by crafting spam messages in specific ways [92, 100]. In general, the drawback of post-acceptance methods is that an email has to be received before it can be analyzed.

Pre-acceptance methods attempt to detect spam before actually receiving the full message. Some analysis techniques take the *origin* of an email into account and analyze distinctive features about the sender of an email (e.g., the IP address or autonomous system the email is sent from, or the geographical distance between the sender and the receiver) [67, 142, 112]. In practice, these sender-based techniques have coverage problems: previous work showed how IP blacklists miss detecting a large fraction of the IP addresses that are actually sending spam, especially due to the highly dynamic nature of the machines that send spam (typically botnets) [119, 109, 125].

In Chapter 3 I show that our system to detect infected hosts that send spam can be effectively used to increase the coverage provided by IP blacklists.

2.2.4 Detecting Malicious Online Service Accounts

To perform their malicious activity, such as spreading malicious content, cybercriminals typically instruct their bots to connect to accounts on legitimate online services that are under their control. To obtain such accounts, they can pay workers to create account for them [145], purchase mass-created fake accounts [138], or buy credentials of compromised accounts on such services [125]. Given the magnitude of the problem, numerous approaches have been proposed to detect accounts that perform malicious activities on online services.

In Chapter 5, I will present SPAMDETECTOR, one of the first systems that was developed to detect fake accounts used to spread spam on social networks. Similar to our system, other early detection techniques for malicious activity on social networks focused on identifying fake accounts and spam messages [32, 87] by leveraging features that are geared towards recognizing characteristics of spam accounts (such as the presence of URLs in messages or the message similarity across user posts). Yu et al. [154] proposed a system to detect fake social network accounts; the system looks at the network structure, and flags accounts that are not well-connected with their peers in the network as possibly malicious. The idea behind this approach is that fake accounts will stand out compared to legitimate ones, because legitimate users will connect less frequently to them than they do with people they know in real life. To overcome this problem, miscreants started connecting several fake accounts with each others, creating dense neighborhoods. Although these connections will make them look more similar to legitimate accounts, fake accounts will still be reasonably isolated from legitimate communities of accounts. Cai et al. [40] proposed a system that detects fake profiles on social networks by examining densely interconnected groups of profiles. These techniques work reasonably well, and both Twitter and Facebook rely on similar heuristics to detect fake accounts [62, 140].

Yang et al. [151] performed an analysis of fake accounts on Renren, a popular Chinese microblogging platform [21]. Their results contradict previous work; in fact, they show that fake accounts on Renren do not typically create connections among each other, but they tend to integrate into the regular social graph.

In response to defense efforts by social network providers, the focus of the attack-

ers has shifted, and a large fraction of the accounts carrying out malicious activities were not created for this purpose, but started as legitimate accounts that were compromised [61, 63]. Since these accounts do not show a consistently malicious behavior, previous systems will fail to recognize them as malicious. Grier et al. [63] studied the behavior of compromised accounts on Twitter by entering the credentials of an account they controlled on a phishing campaign site. They then observed that the account started sending out phishing tweets, and searched Twitter for similar tweets, identifying thousands of compromised accounts. This approach does not scale as it requires identifying and joining each new phishing campaign. Also, this approach is limited to phishing campaigns. In Chapter 6, I will present COMPA, the first system to detect that a social network account has been compromised, and is now used by a cybercriminal to spread malicious content.

Gao et al. [61] developed a clustering approach to detect spam wall posts on Facebook. They also attempted to determine whether an account that sent a spam post was compromised. To this end, the authors look at the wall post history of spam accounts. However, the classification is very simple. When an account received a benign wall post from one of their connections (friends), they automatically considered that account as being legitimate but compromised. The problem with this technique is that, as we show in Chapter 5, spam victims occasionally send messages to these spam accounts. This would cause their approach to detect legitimate accounts as compromised. Moreover, the system needs to know whether an account has sent spam before it can classify it as fake or compromised. COMPA, on the other hand, detects compromised accounts also when they are not involved in spam campaigns. As an improvement to these techniques, Gao et al. [60] proposed a system that groups similar messages posted on social

networks together, and makes a decision about the maliciousness of the messages based on features of the message cluster. Although this system can detect compromised accounts, as well as fake ones, their approach is focused on detecting accounts that spread URLs through their messages and, therefore, is not as generic as COMPA. Other approaches look at how messages propagate on social networks, looking for messages that spread anomalously, such as worms [41, 149].

Lee et al. [88] proposed WARNINGBIRD, a system that detects spam links posted on Twitter by analyzing the characteristics of HTTP redirection chains that lead to a final spam page.

Xu et al. [149] present a system that, by monitoring a small number of nodes, detects worms propagating on social networks. This paper does not directly address the problem of compromised accounts, but could detect large-scale infections such as *koobface* [30]. Chu et al. [49] analyze three categories of Twitter users: humans, bots, and cyborgs, which are software-aided humans that share characteristics from both bots and humans. To this end, the authors use a classifier that examines how regularly an account tweets, as well as other account features such as the application that is used to post updates. Using this paper's terminology, compromised accounts would fall in the cyborg category. However, the paper does not provide a way of reliably detecting them, since these accounts are often times misclassified as either bots or humans. More precisely, their true positive rate for *cyborg* accounts is only of 82.8%. The system that I present in Chapter 6 can detect such accounts much more reliably. Also, the authors in [49] do not provide a clear distinction between compromised accounts and legitimate ones that use third-party applications to post updates on Twitter.

Yang et al. [150] studied new Twitter spammers that act in a stealthy way to avoid detection. In their system, they use advanced features such as the topology of the network that surrounds the spammer. They do not try to distinguish compromised from fake accounts. For their analysis they used the features presented in Chapter 5 as a baseline, and present additional features that are able to detect stealthier accounts used to send spam.

Wang et al. [144] proposed a technique to detect malicious accounts on social networks, based on the sequence of clicks that the people (or the programs) controlling such accounts perform. Jacob et al. [74] presented PUBCRAWL, a system that detects accounts that are used to crawl online services. Benevenuto et al. presented a system to detect accounts that leverage the Youtube service to spread malicious content [33].

In Chapter 7 I present EVILCOHORT, the first system that, instead of detecting malicious hosts or malicious accounts separately, observes the way in which bot-infected machines access online accounts, and detects communities of accounts that are accessed by the same botnet. As I will show, this allows us to flag accounts as malicious although they do not spread any malicious content on the online service, but for example use it as a C&C channel.

Part I

Detecting Misbehaving Hosts

Chapter 3

Locating Spambots on the Internet

3.1 Introduction

While different botnets serve different, nefarious goals, one important purpose of botnets is the distribution of spam emails. Recent studies indicate that, nowadays, about 85% of the overall spam traffic on the Internet is sent with the help of botnets [97, 133]. The reason for this prevalence is that botnets provide two advantages for spammers. First, a botnet serves as a convenient infrastructure for sending out large quantities of messages; it is essentially a large, distributed computing system with massive bandwidth. A botmaster can send out tens of millions of emails within a few hours using thousands of infected machines. Second, a botnet allows an attacker to evade spam filtering techniques based on the sender IP addresses. The reason is that the IP addresses of some infected machines change frequently (e.g., due to the expiration of a

DHCP lease, or to the change in network location in the case of an infected portable computer). Moreover, it is easy to infect machines and recruit them as new members into a botnet. This means that blacklists need to be updated constantly by tracking the IP addresses of spamming bots.

Tracking spambots is challenging. One approach to detect infected machines is to set up *spam traps*. These are fake email addresses (i.e., addresses not associated with real users) that are published throughout the Internet with the purpose of attracting and collecting spam messages. By extracting the sender IP addresses from the emails received by a spam trap, it is possible to obtain a list of bot-infected machines. However, this approach faces two main problems. First, it is likely that only a subset of the bots belonging to a certain botnet will send emails to the spam trap addresses. Therefore, the analysis of the messages collected by the spam trap can provide only a partial view of the activity of the botnet. Second, some botnets might only target users located in a specific country (e.g., due to the language used in the email), and thus a spam trap located in a different country would not observe those bots. Other approaches to identify the hosts that are part of a spamming botnet are specific to particular botnets. For example, by taking control of the command & control (C&C) component of a botnet [102, 110], or by analyzing the communication protocol used by the bots to interact with other components of the infrastructure [44, 84, 121], it is possible to enumerate (a subset of) the IP addresses of the hosts that are part of a botnet. In these cases, however, the results are specific to the particular botnet that is being targeted (and, typically, the type of C&C used). As a result of these problems, blacklists of computers that are known to send spam are incomplete, and often fail in detecting spambots that have recently appeared [109].

In this chapter, we present a novel approach to identify and track spambot populations on the Internet. Our ambitious goal is to track the IP addresses of all active hosts that belong to every spamming botnet. By active hosts, we mean hosts that are online and that participate in spam campaigns. Comprehensive tracking of the IP addresses belonging to spamming botnets is useful for several reasons:

- Internet Service Providers can take countermeasures to prevent the bots whose IP addresses reside in their networks from sending out email messages.
- Organizations can clean up compromised machines in their networks.
- Existing blacklists and systems that analyze network-level features of emails can be improved by providing accurate information about machines that are currently sending out spam emails.
- By monitoring the number of bots that are part of different botnets, it is possible to guide and support mitigation efforts so that the C&C infrastructures of the largest, most aggressive, or fastest-growing botnets are targeted first.

Our approach to tracking spamming bots is based on the following insight: bots that belong to the same botnet share the same C&C infrastructure and the same code base. As a result, these bots will feature similar behavior when sending spam [64, 148, 152]. In contrast, bots belonging to different spamming botnets will typically use different parameters for sending spam mails (e.g., the size of the target email address list, the domains or countries that are targeted, the spam contents, or the timing of their actions). More precisely, we leverage the fact that bots (of a particular botnet) that participate in a *spam campaign* share similarities in the destinations (domains) that they target and

in the time periods they are active. Similar to previous work [84], we consider a spam campaign to be a set of email messages that share a substantial amount of content and structure (e.g., a spam campaign might involve the distribution of messages that promote a specific pharmaceutical scam).

3.1.1 Input Datasets

At a high level, our approach takes two datasets as input. The first dataset contains the IP addresses of known spamming bots that are active during a certain time period (we call this time period the *observation period*). The IP addresses are grouped by spam campaign. That is, IP addresses in the same group sent the same type of messages. We refer to these groups of IP addresses as *seed pools*. The second dataset is a log of email transactions carried out on the Internet during the same time period. This log, called the *transaction log*, contains entries that specify that, at a certain time, IP address C attempted to send an email message to IP address S. The log does not need to be a complete log of every email transaction on the Internet (as it would be unfeasible to collect this information). However, as we will discuss later, our approach becomes more effective as this log becomes more comprehensive.

3.1.2 Approach

In the first step of our approach, we search the transaction log for entries in which the sender IP address is one of the IP addresses in the seed pools (i.e., the known spambots). Then, we analyze these entries and generate a number of behavioral profiles that capture

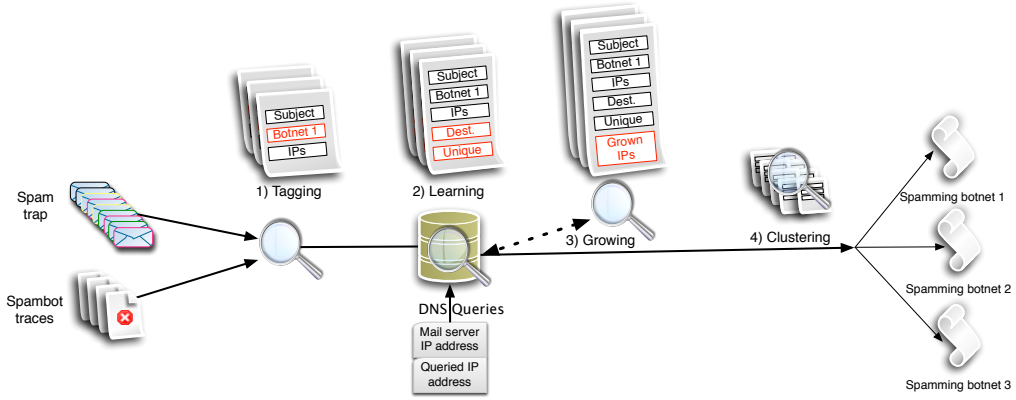


Figure 3.1: Overview of BOTMAGNIFIER.

the way in which the hosts in the seed pools sent emails during the observation period.

In the second step of the approach, the whole transaction log is searched for patterns of behavior that are similar to the spambot behavior previously learned from the seed pools. The hosts that behave in a similar manner are flagged as possible spamming bots, and their IP addresses are added to the corresponding *magnified pool*.

In the third and final step, heuristics are applied to reduce false positives and to assign spam campaigns (and the IP addresses of bots) to specific botnets (e.g., *Rustock* [43], *Cutwail* [125], or *MegaD* [39, 44]).

We implemented our approach in a tool, called BOTMAGNIFIER. In order to populate our seed pools, we used data from a large spam trap set up by an Internet Service Provider (ISP). Our transaction logs were constructed by running a mirror for *Spamhaus* [13], a popular DNS-based blacklist. Note that other sources of information can be used to either populate the seed pools or to build a transaction log. As we will show, BOTMAGNIFIER also works for transaction logs extracted from netflow data

collected from a large ISP's backbone routers.

BOTMAGNIFIER is executed periodically, at the end of each observation period. It outputs a list of the IP addresses of all bots in the magnified pools that were found during the most recent period. Moreover, BOTMAGNIFIER associates with each seed and magnified pool a label that identifies (when possible) the name of the botnet that carried out the corresponding spam campaign. Our experimental results show that our system can find a significant number of additional IP addresses compared to the seed baseline. Furthermore, BOTMAGNIFIER is able to detect emerging spamming botnets. As we will show, we identified the resurrection of the *Waledac* spamming botnet during the evaluation period, demonstrating the ability of our technique to find new botnets. An overview of BOTMAGNIFIER is shown in Figure 3.1.

3.2 Input Datasets

BOTMAGNIFIER requires two input datasets to track spambots: *seed pools* and a *transaction log*. In this section, we discuss how these two datasets are obtained.

3.2.1 Seed Pools

A *seed pool* is a set of IP addresses of hosts that, during the most recent observation period, participated in a specific spam campaign. The underlying assumption is that the hosts whose IP addresses are in the same seed pool are part of the same spamming botnet, and they were instructed to send a certain batch of messages (e.g., emails

advertising cheap Viagra or replica watches).

To generate the seed pools for the various spam campaigns, we took advantage of the information collected by a spam trap set up by a large US ISP. Since the email addresses used in this spam trap do not correspond to real customers, all the received emails are spam. We collected data from the spam trap between September 1, 2010 and February 10, 2011, with a downtime of about 15 days in November 2011. The spam trap collected, on average, 924,000 spam messages from 268,000 IP addresses every day.

Identifying similar messages. We identify spam campaigns within this dataset by looking for similar email messages. More precisely, we analyze the subject lines of all spam messages received during the last observation period (currently one day: see discussion below). Messages that share a similar subject line are considered to be part of the same campaign (during this period).

Unfortunately, the subject lines of messages of a certain campaign are typically not identical. In fact, most botnets vary the subject lines of the message they send to avoid detection by anti-spam systems. For example, some botnets put the user name of the recipient in the subject, or change the price of the pills being sold in drug-related campaigns. To mitigate this problem, we extract *templates* from the actual subject lines. To this end, we substitute user names, email addresses, and numbers with placeholder regular expressions. User names are recognized as tokens that are identical to the first part of the destination email address (the part to the left of the @ sign). For example, the subject line "`john, get 90% discounts!`" sent to user `john@example.com` becomes "`\w+, get [0-9]+% discounts!`"

More sophisticated botnets, such as *Rustock*, add random text fetched from Wikipedia to both the email body and the subject line. Other botnets, such as *Lethic*, add a random word at the end of each subject. These tricks make it harder to group emails belonging to the same campaign that are sent by different bots, because different bots will add unique text to each message. To handle this problem, we developed a set of custom rules for the largest spamming botnets that remove the spurious content from the subject lines.

Once the subjects of the messages have been transformed into templates and the spurious information has been removed, messages with the same template subject line are clustered together. This approach is less sophisticated than methods that take into account more features of the spam messages [104, 148], but we found (by manual investigation) that our simple approach was very effective for our purpose. Our approach, although sufficient, could be refined even further by incorporating these more sophisticated schemes to improve our ability to recognize spam campaigns.

Once the messages are clustered, the IP addresses of the senders in each cluster are extracted. These sets of IP addresses represent the seed pools that are used as input to our magnification technique.

Seed pool size. During our experiments, we found that seed pools that contain a very small number of IP addresses do not provide good results. The reason is that the behavior patterns that can be constructed from only a few known bot instances are not precise enough to represent the activity of a botnet. For example, campaigns involving 200 unique IP addresses in the seed pool produced, on average, magnified sets where 60%

of the IP addresses were not listed in *Spamhaus*, and therefore were likely legitimate servers. Similarly, campaigns with a seed pool size of 500 IP addresses still produced magnified sets where 25% of the IP addresses were marked as legitimate by *Spamhaus*. For these reasons, we only consider those campaigns for which we have observed more than 1,000 unique sender IP addresses. The emails belonging to these campaigns account for roughly 84% of the overall traffic observed by our spam trap. It is interesting to notice that 8% of the overall traffic belongs to campaigns carried out by less than 10 distinct IP addresses per day. Such campaigns are carried out by dedicated servers and abused email service providers. The aggressive spam behavior of these servers and their lack of geographic/IP diversity makes them trivial to detect without the need for magnification.

The lower limit on the size of seed pools has implications for the length of the observation period. When this interval is too short, the seed pools are likely to be too small. On the other hand, many campaigns last less than a few hours. Thus, it is not useful to make the observation period too long. Also, when increasing the length of the observation period, there is a delay introduced before BOTMAGNIFIER can identify new spam hosts. This is not desirable when the output is used for improving spam defenses. In practice, we found that an observation period of one day allows us to generate sufficiently large seed pools from the available spam feed. To evaluate the impact that the choice of the analysis period might have on our analysis system, we looked at the duration of 100 spam campaigns, detected over a period of one day. The average duration of these campaigns is 9 hours, with a standard deviation of 6 hours. Of the campaigns that we analyzed, 25 lasted less than four hours. However, only two of these campaigns did not generate large enough seed pools to be considered by BOTMAGNIFIER. On the

other hand, 8 campaigns that lasted more than 18 hours would not have generated large enough seed pools if we used a shorter observation period. Also, by manual investigation, we found that campaigns that last more than one day typically reach the threshold of 1,000 IP addresses for their seed pool within the first day. Therefore, we believe that the choice of an observation period of one day works well, given the characteristics of the transaction log we used. Of course, if the volume of either the seed pools or the transaction log increased, the observation period could be reduced accordingly, making the system more effective for real-time spam blacklisting.

Note that it is not a problem when a spam campaign spans multiple observation periods. In this case, the bots that participate in this spam campaign and are active during multiple periods are simply included in multiple seed pools, one for each observation period, for this campaign.

3.2.2 Transaction Log

The transaction log is a record of email transactions carried out on the Internet during the same time period used for the generation of the seed pools. For the current version of BOTMAGNIFIER and the majority of our experiments, we obtained the transaction log by analyzing the queries to a mirror of *Spamhaus*, a widely-used DNS-based blacklisting service (DNSBL). When an email server S is contacted by a client C that wants to send an email message, server S contacts one of the *Spamhaus* mirrors and asks whether the IP address of the client C is a known spamming host. If C is a known spammer, the connection is closed before the actual email is sent.

Each query to *Spamhaus* contains the IP address of C. It is possible that S may not query *Spamhaus* directly. In some cases, S is configured to use a local DNS server that forwards the query. In such cases, we would mistakenly consider the IP address of the DNS server as the mail server. However, the actual value of the IP address of S is not important for the subsequent analysis. It is only important to recognize when two different clients send email to the *same* server S. Thus, as long as emails sent to server S yield *Spamhaus* queries that always come from the same IP address, our technique is not affected.

Each query generates an entry in the transaction log. More precisely, the entry contains a timestamp, the IP address of the sender of the message, and the IP address of the server issuing the query. Of course, by monitoring a single *Spamhaus* mirror (out of 60 deployed throughout the Internet), we can observe only a small fraction of the global email transactions. Our mirror observes roughly one hundred million email transactions a day, compared to estimates that put the number of emails sent daily at hundreds of billions [79].

Note that even though *Spamhaus* is a blacklisting service, we do not use the information it provides about the blacklisted hosts to perform our analysis. Instead, we use the *Spamhaus* mirror only to collect the transaction logs, regardless of the fact that a sender may be a known spammer. In fact, other sources of information can be used to either populate the seed pools or to collect the transaction log. To demonstrate this, we also ran BOTMAGNIFIER on transaction logs extracted from netflow data collected from a number of backbone routers of a large ISP. The results show that our general approach is still valid (see Section 3.6.4 for details).

3.3 Characterizing Bot Behavior

Given the two input datasets described in the previous section, the first step of our approach is to extract the behavior of known spambots. To this end, the transaction log is consulted. More precisely, for each seed pool, we query the transaction log to find all events that are associated with all of the IP addresses in that seed pool (recall that the IP addresses in a seed pool correspond to known spambots). Here, an event is an entry in the transaction log where the known spambot is the *sender* of an email. Essentially, we extract all the instances in the transaction log where a known bot has sent an email.

Once the transaction log entries associated with a seed pool are extracted, we analyze the *destinations* of the spam messages to characterize the bots' behavior. That is, the behavior of the bots in a seed pool is characterized by the set of destination IP addresses that received spam messages. We call the set of server IP addresses targeted by the bots in a seed pool this pool's *target set*.

The reason for extracting a seed pool's target set is the insight that bots belonging to the same botnet receive the same list of email addresses to spam, or, at least, a subset of addresses belonging to the same list. Therefore, during their spamming activity, bots belonging to botnet A will target the addresses contained in list L_A, while bots belonging to botnet B will target destinations belonging to list L_B. That is, the targets of a spam campaign characterize the activity of a botnet. This observations has been confirmed by previous work [73, 125].

Unfortunately, the target sets of two botnets often have substantial overlap. The reason is that there are many popular destinations (server addresses) that are targeted by most

botnets (e.g., the email servers of Google, Yahoo, large ISPs with many users, etc.) Therefore, we want to derive, for each spam campaign (seed pool), the most *characterizing set* of destination IP addresses. To this end, we remove from each pool's target set all server IP addresses that appear in any target set belonging to another another seed pool during that observation period.

More precisely, consider the seed pools $P = p_1, p_2, \ldots, p_n$. Each pool p_i stores the IP addresses of known bots that participated in a certain campaign: $i_1, i_2, \ldots, i_m$. In addition, consider that the transaction log L contains entries in the form $\langle t, i_s, i_d \rangle$, where t is a time stamp, i_s is the IP address of the sender of an email and i_d is the IP address of the destination server of an email. For each seed pool p_i, we build this seed pool's target set $T(p_i)$ as follows:

$$T(p_i) := \{i_d | \langle t, i_s, i_d \rangle \in L \wedge i_s \in p_i\}. \tag{3.1}$$

Then, we compute the characterizing set $C(p_i)$ of a seed pool p_i as follows:

$$C(p_i) := \{i_d | i_d \in T(p_i) \wedge i_d \notin T(p_j), j \neq i\}. \tag{3.2}$$

As a result, $C(p_i)$ contains only the target addresses that are unique (characteristic) for the destinations of bots in seed pool p_i. The characterizing set $C(p_i)$ of each pool is the input to the next step of our approach.

3.4 Bot Magnification

The goal of the bot magnification step is to find the IP addresses of additional, previously-unknown bots that have participated in a known spam campaign. More precisely, the goal of this step is to search the transaction log for IP addresses that behave similarly to the bots in a seed pool p_i. If such matches can be found, the corresponding IP addresses are added to the *magnification set* associated with p_i. This means that a magnification set stores the IP addresses of additional, previously-unknown bots.

BOTMAGNIFIER considers an IP address x_i that appears in the transaction log L as matching the behavior of a certain seed pool p_i (and, thus, belonging to that spam campaign) if the following three conditions hold:

1. Host x_i sent emails to at least N destinations in the seed pool's target set $T(p_i)$.
2. Host x_i never sent an email to a destination that does *not* belong to that target set.
3. Host x_i has contacted *at least one* destination that is unique for seed pool p_i (i.e., an address in $C(p_i)$).

If all three conditions are met, then IP address x_i is added to the magnification set $M(p_i)$ of seed pool p_i.

More formally, if we define $D(x_i)$ as the set of destinations targeted by an IP address x_i, we have:

$$\begin{aligned} x_i \in M(p_i) \quad \Longleftrightarrow \quad & |D(x_i) \cap T(p_i)| \geq N \wedge \\ & D(x_i) \subseteq T(p_i) \wedge \\ & D(x_i) \cap C(p_i) \neq \emptyset. \end{aligned} \tag{3.3}$$

The intuition behind this approach is the following: when a host h sends a reasonably large number of emails to the same destinations that were targeted by a spam campaign and not to any other targets, there is a strong indication that the email activity of this host is similar to the bots involved in the campaign. Moreover, to assign a host h to at most one campaign (the one that it is most similar), we require that h targets at least one unique destination of this campaign.

3.4.1 Threshold Computation

The main challenge in this step is to determine an appropriate value for the threshold N, which captures the minimum number of destination IP addresses in $T(p_i)$ that a host must send emails to in order to be added to the magnification set $M(p_i)$. Setting N to a value that is too low will generate too many bot candidates, including legitimate email servers, and the tool would generate many false positives. Setting N to a value that is too high might discard many bots that should have been included in the magnification set (that is, the approach generates many false negatives). This trade-off between false

positives and false negatives is a problem that appears in many security contexts, for example, when building models for intrusion detection.

An additional, important consideration for the proper choice of N is the size of the target set $|T(p_i)|$. Intuitively, we expect that N should be larger when the size of the target set increases. This is because a larger target set increases the chance that a random, legitimate email sender hits a sufficient number of targets by accident, and hence, will be incorrectly included into the magnification set. In contrast, bots carrying out a spam campaign that targets only a small number of destinations are easier to detect. The reason is that as soon as a legitimate email sender sends an email to a server that is *not* in the set targeted by the campaign, it will be immediately discarded by our magnification algorithm. Therefore, we represent the relationship between the threshold N and the size of the target set $|T(p_i)|$ as:

$$N = k \cdot |T(p_i)|, 0 < k \leq 1, \tag{3.4}$$

where k is a parameter. Ideally, the relation between N and $|T(p_i)|$ would be linear, and k will have a constant value. However, as will be clear from the discussion below, k also varies with the size of $|T(p_i)|$.

To determine a good value for k and, as a consequence, select a proper threshold N, we performed an analysis based on ground truth about the actual IP addresses involved in several spam campaigns. This information was collected from the takedown of more than a dozen C&C servers used by the Cutwail spam botnet. More specifically, each server stored comprehensive records (e.g., target email lists, bot IP addresses, etc.)

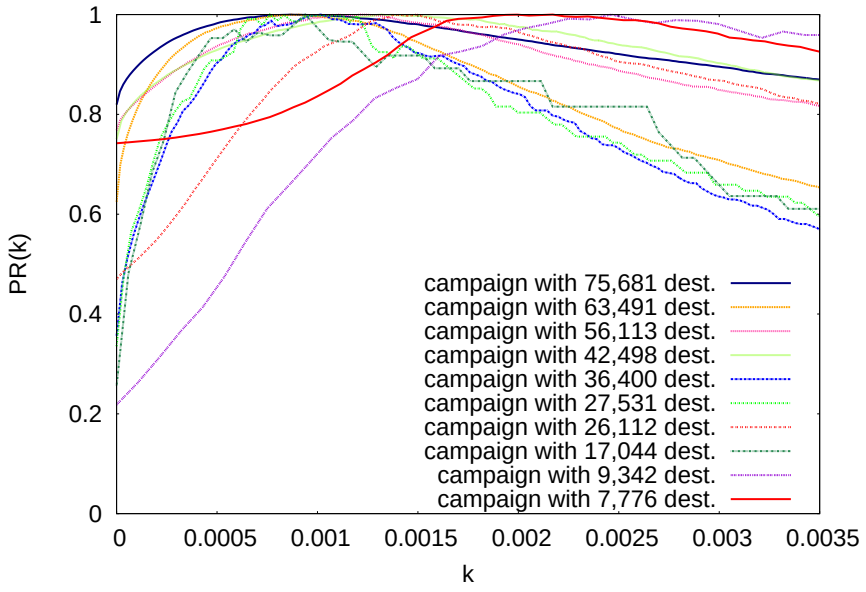


Figure 3.2: Quality of magnification for varying k using ten *Cutwail* campaigns of different sizes.

about spam activities for a number of different campaigns [125].

In particular, we applied BOTMAGNIFIER to ten *Cutwail* campaigns, extracted from two different C&C servers. We used these ten campaigns since we had a precise view of the IP addresses of the bots that sent the emails. For the experiment, we varied the value for N in the magnification process from 0 to 300. This analysis yielded different magnification sets for each campaign. Then, using our knowledge about the actual bots B that were part of each campaign, we computed the precision P and recall R values for each threshold setting. Since we want to express the quality of the magnification process as a function of k, independently of the size of a campaign, we use Equation 3.4 to get $k = \frac{N}{|T(p_i)|}$.

The precision value $P(k)$ represents what fraction of the IP addresses that we obtain

as candidates for the magnification set for a given k are actually among the ground truth IP addresses. The recall value $R(k)$, on the other hand, tells us what fraction of the total bot set B is identified. Intuitively, a low value of k will produce high $R(k)$, but low $P(k)$. When we increase k, $P(k)$ will increase, but $R(k)$ will decrease. Optimally, both precision and recall are high. Thus, for our analysis, we use the product $PR(k) = P(k) \cdot R(k)$ to characterize the quality of the magnification step. Figure 3.2 shows how $PR(k)$ varies for different values of k. As shown for each campaign, $PR(k)$ first increases, then stays relatively level, and then starts to decrease.

The results indicate that k is not a constant, but varies with the size of $|T(p_i)|$. In particular, small campaigns have a higher optimal value for k compared to larger campaigns: as $|T(p_i)|$ increases, the value of k slowly decreases. To reflect this observation, we use the following, simple way to compute k:

$$k = k_b + \frac{\alpha}{|T(p_i)|}, \tag{3.5}$$

where k_b is a constant value, α is a parameter, and $|T(p_i)|$ is the number of destinations that a campaign targeted. The parameters k_b and α are determined so that the quality of the magnification step PR is maximized for a given ground truth dataset. Using the *Cutwail* campaigns as the dataset, this yields $k_b = 8 \cdot 10^{-4}$ and $\alpha = 10$.

Our experimental results show that these parameter settings yield good results for a wide range of campaigns, carried out by several different botnets. This is because the magnification process is robust and not dependent on an optimal threshold selection. We found that non-optimal thresholds typically tend to decrease recall. That is, the

magnification process does not find all bots that it could possibly detect, but false positives are limited. In Section 3.6.4, we show how the equation of k, with the values we determined for parameters k_b and α, yields good results for any campaign magnified from our *Spamhaus* dataset. We also show that the computation of k can be performed in the same way for different types of transaction logs. To this end, we study how BOTMAGNIFIER can be used to analyze netflow records.

3.5 Spam Attribution

Once the magnification process has completed, we merge the IP addresses from the seed pool and the magnification set to obtain a *campaign set*. We then apply several heuristics to reduce false positives and to assign the different campaign sets to specific botnets. Note that the labeling of the campaign sets does not affect the results of the bot magnification process. BOTMAGNIFIER could be used in the wild for bot detection without these attribution functionalities. It is relevant only for tracking the populations of known botnets, as we discuss in Section 3.6.2.

3.5.1 Spambot Analysis Environment

The goal of this phase is to understand the behavior of current spamming botnets. That is, we want to determine the types of spam messages sent by a specific botnet at a certain point in time. To this end, we have built an environment that enables us to execute bot binaries in a controlled setup similarly to previous studies [76, 147].

Our spambot analysis environment is composed of one physical system hosting several virtual machines (VMs), each of which executes one bot binary. The VMs have full network access so that the bots can connect to the C&C server and receive spam-related configuration data, such as spam templates or batches of email addresses to which spam should be sent. However, we make sure that no actual spam emails are sent out by sinkholing spam traffic, i.e., we redirect outgoing emails to a mail server under our control. This server is configured to record the messages, without relaying them to the actual destination. We also prevent other kinds of malicious traffic (e.g., scanning or exploitation attempts) through various firewall rules. Some botnets (e.g., *MegaD*) use TCP port 25 for C&C traffic, and, therefore, we need to make sure that such bots can still access the C&C server. This is implemented by firewall rules that allow C&C traffic through, but prevent outgoing spam. Furthermore, botnets such as *Rustock* detect the presence of a virtualization environment and refuse to run. Such samples are executed on a physical machine configured with the same network restrictions. To study whether bots located in different countries show a unique behavior, we run each sample at two distinct locations: one analysis environment is located in the United States, while the other one is located in Europe. In our experience, this setup enables us to reliably execute known spambots and observe their current spamming behavior.

For this study, we analyzed the five different bot families that were the most active during the time of our experiments: *Rustock* [43], *Lethic*, *MegaD* [39, 44], *Cutwail* [125], and *Waledac* [102]. We ran our samples from July 2010 to February 2011. Some of the spambots we ran sent out spam emails for a limited amount of time (typically, a couple of weeks), and then lost contact with their controllers. We periodically substituted such bots with newer samples. Other bots (e.g., *Rustock*) were active for most of the analysis

period.

3.5.2 Botnet Tags

After monitoring the spambots in a controlled environment, we attempt to assign botnet labels to spam emails found in our spam trap. Therefore, we first extract the subject templates from the emails that were collected in the analysis environment with the same technique described in Section 3.2.1. Then, we compare the subject templates with the emails we received in the spam trap during that same day. If we find a match, we tag the campaign set that contains the IP address of the bot that sent the message with the corresponding botnet name. Otherwise, we keep the campaign set unlabeled.

3.5.3 Botnet Clustering

As noted above, we ran five spambot families in our analysis environment. Of course, it is possible that one of the monitored botnets is carrying out more campaigns than those observed by analyzing the emails sent by the bots we execute in our analysis environment. In addition, we are limited by the fact that we cannot run all bot binaries in the general case (e.g., due to newly emerging botnets or in cases where we do not have access to a sample), and, thus, we cannot collect information about such campaigns. The overall effect of this limitation is that some campaign sets may be left unlabeled.

The goal of the botnet clustering phase is to determine whether an unlabeled campaign set belongs to one of the botnets we monitored. If an unlabeled campaign set cannot be

associated with one of the existing labeled campaign sets, then we try to see if it can be merged with another unlabeled campaign set, which, together, might represent a new botnet.

In both cases, there is a need to determine if two campaign sets are "close" enough to each other in order to be considered as part of the same botnet. In order to represent the distance between campaign sets, we developed three metrics, namely an IP overlap metric, a destination distance metric, and a bot distance metric.

IP overlap. The observation underlying the IP overlap metric is that two campaign sets sharing a large number of bots (i.e., common IP addresses) likely belong to the same botnet. It is important to note that infected machines can belong to multiple botnets, as one machine may be infected with two distinct instances of malware. Another factor that we need to take into account is network address translation (NAT) gateways, which can potentially hide large networks behind them. As a result, the IP address of a NAT gateway might appear as part of multiple botnets. However, a host is discarded from the campaign set related to p_i as soon as it contacts a destination that is *not* in the target set (see Section 3.4 for a discussion). Therefore, NAT gateways are likely to be discarded from the candidate set early on: at some point, machines behind the NAT will likely hit two destinations that are unique to two different seed pools, and, thus, will be discarded from all campaign sets. This might not be true for small NATs, with just a few hosts behind them. In this case, the IP address of the gateway would be detected as a bot by BOTMAGNIFIER. In a real world scenario, this would still be useful information for the network administrator, who would know what malware has

likely infected one or more of her hosts.

Given these assumptions, we merge two campaign sets with a large IP overlap. More precisely, first the intersection of the two campaign sets is computed. Then, if such intersection represents a sufficiently high portion of the IP addresses in *either* of the campaign sets, the two campaign sets are merged.

The fraction of IP addresses that need to match either of the campaign sets to consider them to be part of the same botnet varies with the size of the sets for those campaigns. Intuitively, two small campaigns will have to overlap by a larger percentage than two large campaigns in order to be considered as part of the same botnet. This is done to avoid merging small campaigns together just based on a small number of IP addresses that might be caused by multiple infections or by two different spambots hiding behind a small NAT. Given a campaign c, the fraction of IP addresses that has to overlap with another campaign in order to be merged together is

$$O_c = \frac{1}{\log_{10}(N_c)}, \tag{3.6}$$

where N_c is the number of hosts in the campaign set. We selected this equation because the denominator increases slowly with the number of bots carrying out a campaign. Moreover, because of the use of the logarithm, this equation models an exponential decay, which decreases fast for small values of N_c, and much more slowly for large values of it. Applying this equation, a campaign carried out by 100 hosts will require an overlap of 50% or more to be merged with another one, while a campaign carried out by 10,000 hosts will only require an overlap of 25%. When comparing two campaigns

c_1 and c_2, we require the smaller one to have an overlap of at least O_c with the largest one to consider them as being carried out by the same botnet.

Destination distance. This technique is an extension of our magnification step. We assume that bots carrying out the same campaign will target the same destinations. However, as mentioned previously, some botnets send spam only to specific countries during a given time frame. Leveraging this observation, it is possible to find out whether two campaign sets are likely carried out by the same botnet by observing the country distribution of the set of destinations they targeted. More precisely, we build a *destination country vector* for each campaign set. Each element of the destination country vector corresponds to the fraction of destinations that belong to a specific country. We determined the country of each IP address using the GEOIP tool [95]. Then, for each pair of campaign sets, we calculate the *cosine distance* between them.

We performed a *precision* versus *recall* analysis to develop an optimal threshold for this clustering technique. By precision, we mean how well this technique can discriminate between campaigns belonging to different botnets. By recall, we capture how well the technique can cluster together campaigns carried out by the same botnet. We ran our analysis on 50 manually-labeled campaigns picked from the ones sent by the spambots in our analysis environment. Similarly to how we found the optimal value of k in Section 3.4, we multiply precision and recall together. We then searched for the threshold value that maximizes this product. In our experiments, we found that the cosine distance of the destination countries vectors is rarely lower than 0.8. This occurs regardless of the particular country distribution of a campaign, because there will be

a significant amount of bots in large countries (e.g., the United States or India). The *precision* versus *recall* analysis showed that 0.95 is a good threshold for this clustering technique.

Bot distance. This technique is similar to the destination distance, except that it utilizes the country distribution of the bot population of the campaign set instead of the location of the targeted servers. For each campaign set, we build a *source country vector* that contains the fraction of bots for a given country.

The intuition behind this technique comes from the fact that malware frequently propagates through malicious web sites, or through legitimate web servers that have been compromised [108, 124]. These sites will not have a uniform distribution of users (e.g., a Spanish web site will mostly have visitors from Spanish-speaking countries) and, therefore, the distribution of compromised users in the world for that site will not be uniform. For this technique, we also performed a *precision* versus *recall* analysis, in the same way as for the destination distance technique. Again, we experimentally found the optimal threshold to be 0.95.

3.6 Evaluation

To demonstrate the validity of our approach, we first examined the results generated by BOTMAGNIFIER when magnifying the population of a large spamming botnet for which we have ground truth knowledge (i.e., we know which IP addresses belong to the botnet). Then, we ran the system for a period of four months on a large set of real-world

data, and we successfully tracked the evolution of large botnets.

3.6.1 Validation of the Approach

To validate our approach, we studied a botnet for which we had direct data about the number and IP addresses of the infected machines. More precisely, in August 2010, we obtained access to thirteen C&C servers belonging to the *Cutwail* botnet [125]. Note that we only used nine of them for this evaluation, since two had already been used to derive the optimal value of N in Section 3.4, and two were not actively sending spam at the time of the takedown. As discussed before, these C&C servers contained detailed information about the infected machines belonging to the botnet and the spam campaigns carried out. The whole botnet was composed of 30 C&C servers. By analyzing the data on the C&C servers we had access to we found that, during the last day of operation, 188,159 bots contacted these nine servers. Of these, 37,914 ($\approx 20\%$) contacted multiple servers. On average, each server controlled 20,897 bots at the time of the takedown, with a standard deviation of 5,478. Based on these statistics, the servers to which we had access managed the operations of between 29% and 37% of the entire botnet. We believe the actual percentage of the botnet controlled by these servers was close to 30%, since all the servers except one were contacted by more than 19,000 bots during the last day of operation. Only a single server was controlling less than 10,000 bots. Therefore, it is safe to assume that the vast majority of the command and control servers were controlling a similar amount of bots ($\approx$ 20,000 each).

We ran the validation experiment for the period between July 28 and August 16, 2010.

For each of the 18 days, we first selected a subset of the IP addresses referenced by the nine C&C servers. As a second step, with the help of the spam trap, we identified which campaigns had been carried out by these IP address during that day. Then, we generated seed and magnified pools. Finally, we compared the output magnification sets against the ground truth (i.e., the other IP addresses referenced by the C&C servers) to assess the quality of the results.

Overall, BOTMAGNIFIER identified 144,317 IP addresses as *Cutwail* candidates in the campaign set. Of these, 33,550 ($\approx$ 23%) were actually listed in the C&C servers' databases as bots. This percentage is close to the fraction of the botnet we had access to (since we considered 9 out of 30 C&C servers), and, thus, this result suggests that the magnified population identified by our system is consistent. To perform a more precise analysis, we ran BOTMAGNIFIER and studied the magnified pools that were given as an output on a daily basis. The average size of the magnified pools was 4,098 per day. In total, during the 18 days of the experiment, we grew the bot population by 73,772 IP addresses. Of the IP addresses detected by our tool, 17,288 also appeared in the spam trap during at least one other day of our experiment, sending emails belonging to the same campaigns carried out by the C&C servers. This confirms that they were actually *Cutwail* bots. In particular, 3,381 of them were detected by BOTMAGNIFIER *before* they ever appeared in the spam trap, which demonstrates that we can use our system to detect bots before they even hit our spam trap.

For further validation, we checked our results against the *Spamhaus* database, to see if the IP addresses we identified as bots were listed as known spammers or not. 81% of them were listed in the blacklist.

We then tried to evaluate how many of the remaining 27,421 IP addresses were false positives. To do this, we used two techniques. First, we tried to connect to the host to check whether it was a legitimate server. Legitimate SMTP or DNS servers can show up in queries on *Spamhaus* due to several reasons (e.g., in cases where reputation services collect information about sender IP addresses or if an email server is configured to query the local DNS server). Therefore, we tried to determine if an IP address that was not blacklisted at the time of the experiment was a legitimate email or DNS server by connecting to port 25 TCP and 53 UDP. If the server responded, we considered it to be a false positive. Unfortunately, due to firewall rules, NAT gateways, or network policies, some servers might not respond to our probes. For this reason, as a second technique, we executed a reverse DNS lookup on the IP addresses, looking for evidence showing that the host was a legitimate server. In particular, we looked for strings that are typical for mail servers in the hostname. These strings are *smtp*, *mail*, *mx*, *post*, and *mta*. We built this list by manually looking at the reverse DNS lookups of the IP address that were not blacklisted by *Spamhaus*. If the reverse lookup matched one of these strings, we considered the IP address as a legitimate server, i.e., a false positive. In total, 2,845 IP addresses resulted in legitimate servers (1,712 SMTP servers and 1,431 DNS servers), which is 3.8% of the overall magnified population.

We then tried to determine what coverage of the entire *Cutwail* botnet our approach produced. Based on the number of active IP addresses per day we saw on the C&C servers, we estimated that the size of the botnet at the time of the takedown was between 300,000 and 400,000 bots. This means that, during our experiment, we were able to track between 35 and 48 percent of the botnet. Given the limitations of our transaction log (see Section 3.6.2), this is a good result, which could be improved by getting access

to multiple *Spamhaus* servers or more complete data streams.

3.6.2 Tracking Bot Populations

To demonstrate the practical feasibility of our approach, we used BOTMAGNIFIER to track bot populations in the wild for a period of four months. In particular, we ran the system for 114 days between September 28, 2010 and February 5, 2011. We had a downtime of about 15 days in November 2011, during which the emails of the spam trap could not be delivered.

By using our magnification algorithm, our system identified and tracked 2,031,110 bot IP addresses during the evaluation period. Of these, 925,978 IP addresses ($\approx 45.6\%$) belonged to magnification sets (i.e., they were generated by the magnification process), while 1,105,132 belonged to seed pools generated with the help of the spam trap.

Data Streams Limitations

The limited view we have from the transaction log generated by only one DNSBL mirror limits the number of bots we can track each day. BOTMAGNIFIER requires an IP address to appear a minimum number of times in the transaction log, in order to be considered as a potential bot. From our DNSBL mirror, we observed that a medium size campaign targets about 50,000 different destination servers (i.e., $|T(p_i)|$ = 50,000). The value of N for such a campaign, calculated using Equation 3.5, is 50. On an average day, our DNSBL mirror logs activity performed by approximately 4.7 million

mail senders. Of these, only about 530,000 ($\approx 11\%$) appear at least 50 times. Thus, we have to discard a large number of potential bots *a priori*, because of the limited number of transactions our *Spamhaus* mirror observes. If we had access to more transaction logs, our visibility would increase, and, thus, the results would improve accordingly.

Botnet	**Total # of IP addresses**	**# of dynamic IP addresses**	**# of static IP addresses**	**# of events per bot** (per day)
Lethic	887,852	770,517	117,335	101
Rustock	676,905	572,445	104,460	173
Cutwail	319,355	285,223	34,132	208
MegaD	68,117	65,062	3,055	112
Waledac	36,058	32,602	3,450	140

Table 3.1: Overview of the BOTMAGNIFIER results.

Overview of Tracking Results

For each day of analysis, BOTMAGNIFIER identified the largest spam campaigns active during that day (Section 3.2), learned the behavior of a subset of IP addresses carrying out those campaigns (Section 3.3), and grew a population of IP addresses behaving in the same way (Section 3.4). This provided us with the ability to track the population of the largest botnets, monitoring how active they were, and determining which periods they were silent.

A challenging aspect of tracking botnets with BOTMAGNIFIER has been assigning the right label to the various spam campaigns (i.e., the name of the botnet that generated them). Tagging the campaigns that we observed in our honeypot environment was trivial, while for the others we used the clustering techniques described in Section 3.5. In total, we observed 1,475 spam campaigns. We tried to assign a botnet label to each cluster, and every time two clusters were assigned the same label, we merged them

together. After this process, we obtained 38 clusters. Seven of them were large botnets, which generated 50,000 or more bot IP addresses in our magnification results. The others were either smaller botnets, campaigns carried out by dedicated servers (i.e., not carried out by botnets), or errors produced by the clustering process.

We could not assign a cluster to 107 campaigns ($\approx$ 7% of all campaigns), and we magnified these campaigns independently from the others. Altogether, the magnified sets of these campaigns accounted for 20,675 IP addresses ($\approx$ 2% of the total magnified hosts). We then studied the evolution over time and the spamming capabilities of the botnets that we were able to label.

Analysis of Magnification Results

Table 3.1 shows some results from our tracking. For each botnet, we list the number of IP addresses we obtained from the magnification process. Interestingly, *Lethic*, with 887,852 IP addresses, was the largest botnet we found. This result is in contrast with the common belief in the security community that, at the time of our experiment, *Rustock* was the largest botnet [94]. However, from our observation, *Rustock* bots appeared to be more aggressive in spamming than the *Lethic* bots. In fact, each *Rustock* bot appeared, on average, 173 times per day on our DNSBL mirror logs, whereas each *Lethic* bot showed up only 101 times.

For each botnet population we grew, we distinguished between static and dynamic IP addresses. We considered an IP address as dynamic if, during the testing period, we observed that IP address only once. On the other hand, if we observed the same IP ad-

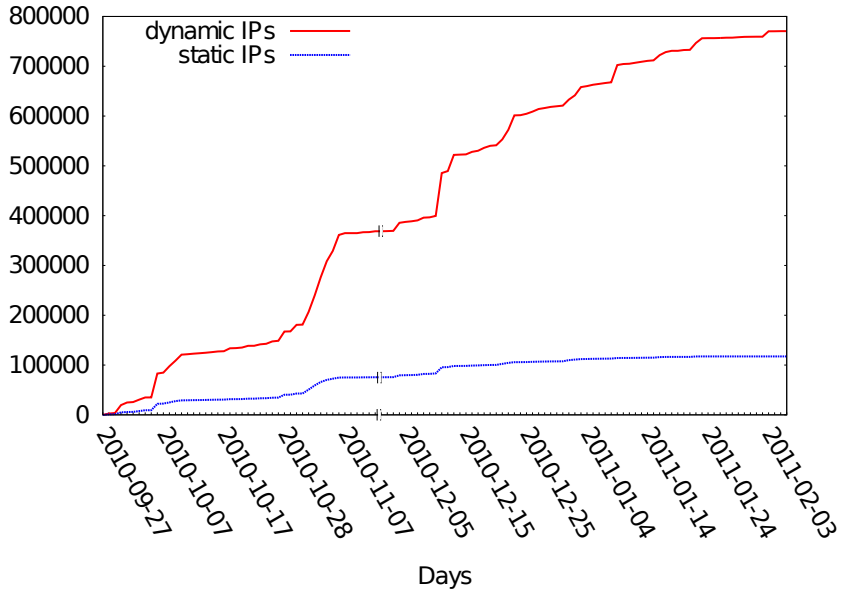


Figure 3.3: Growth of *Lethic* IP addresses.

dress multiple times, we consider it as static. The fraction of static versus dynamic IP addresses for the botnets we tracked goes from 15% for *Rustock* to 4% for *MegaD*. Note that smaller botnets exceeded the campaign size thresholds required by BOTMAGNIFIER (see Section 3.5) less often than larger botnets, and therefore it is possible that our system underestimates the number of IP addresses belonging to the *MegaD* and *Waledac* botnets.

Figures 3.3 and 3.4 show the growth of IP addresses over time for the magnification sets belonging to *Lethic* and *Rustock* (note that we experienced a downtime of the system during November 2010). The figures show that dynamic IP addresses steadily grow over time, while static IP addresses reach saturation after some time. Furthermore, it is interesting to notice that we did not observe much *Rustock* activity between December 24, 2010 and January 10, 2011. Several sources reported that the botnet was (almost)

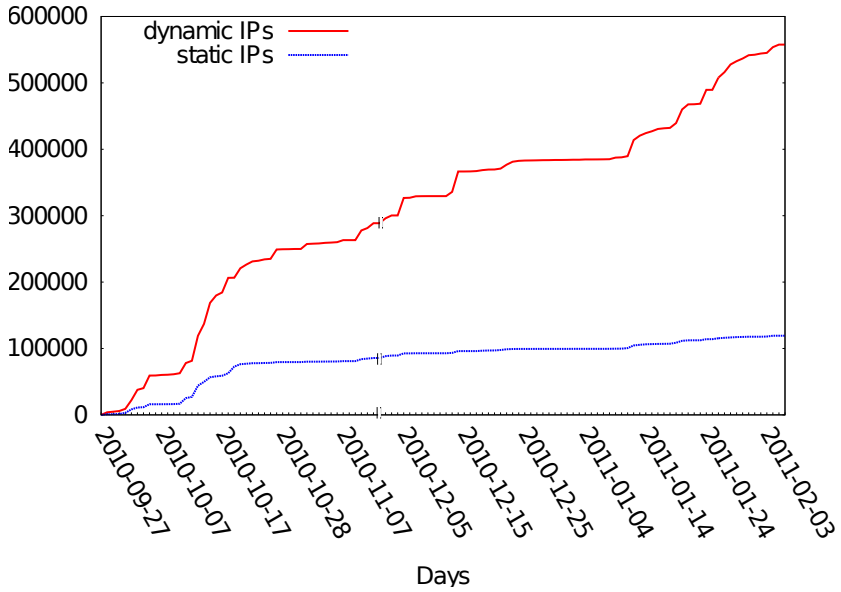


Figure 3.4: Growth of the dynamic and static IP address populations for the two major botnets.

down during this period [83, 134]. BOTMAGNIFIER confirms this downtime of the botnet, which indicates that our approach can effectively track the activity of botnets. After the botnet went back up again in January 2011, we observed a steady growth in the number of *Rustock* IP addresses detected by BOTMAGNIFIER.

Figures 3.5 and 3.6 show the cumulative distribution functions of dynamic IP addresses and static IP addresses tracked during our experiment for the five largest botnets. It is interesting to see that we started observing campaigns carried out by *Waledac* on January 1, 2011. This is consistent with the reports from several sources, who also noticed that a new botnet appeared at the same time [90, 117]. We also observed minimal spam activities associated with *MegaD* after December 7, 2011. This was a few days after the botmaster was arrested [115].

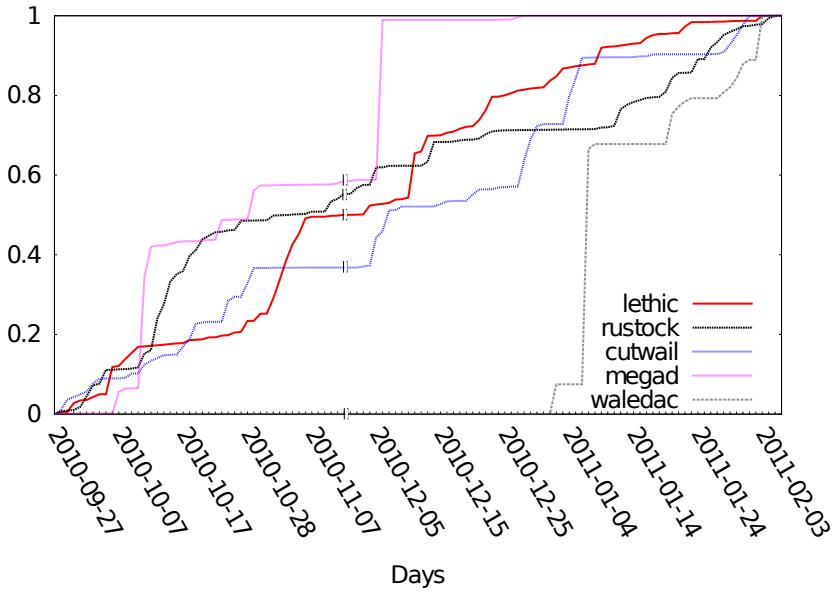


Figure 3.5: Cumulative Distribution Function (CDF) for the dynamic IP addresses.

3.6.3 Application of Results

False positives. In Section 3.4, we showed how the parameter k minimizes the ratio between true positives and false positives. We initially tolerated a small number of false positives because these do not affect the big picture of tracking large botnet populations. However, we want to quantify the false positive rate of the results, i.e., how many of the bot candidates are actually legitimate machines. This information is important, especially if BOTMAGNIFIER is used to inform Internet Service Providers or other organizations about infected machines. Furthermore, if we want to use the results to improve spam filtering systems, we need to be very careful about which IP addresses we consider as bots. We use the same techniques outlined in Section 3.6.1 to check for false positives. We remove each IP address that matches any of these techniques from the magnified sets.

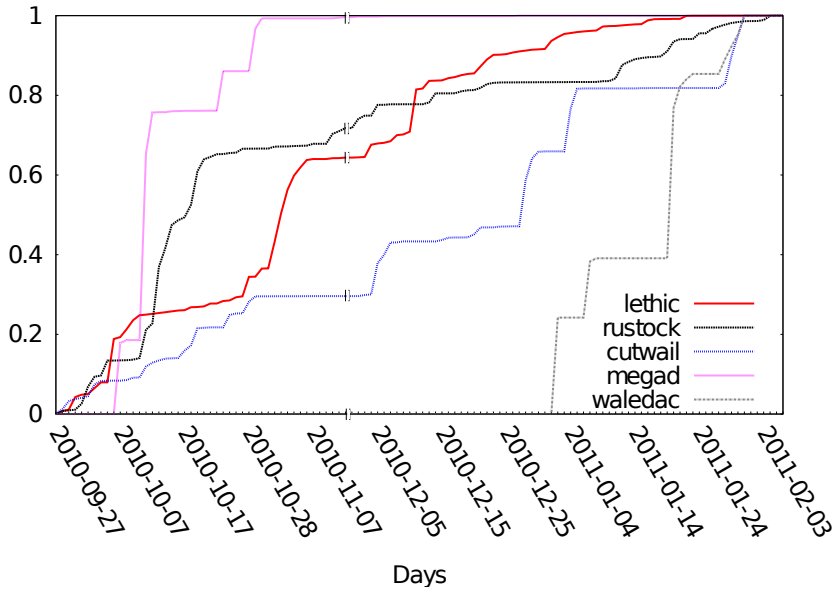


Figure 3.6: Cumulative Distribution Function (CDF) for the static IP addresses.

We ran this false positive detection heuristic on all the magnified IP addresses identified during the evaluation period. This resulted in 35,680 (≈1.6% of the total) IP addresses marked as potential false positives. While this might sound high at first, we also need to evaluate how relevant this false positive rate is in practice: our results can be used to augment existing systems and thus we can tolerate a certain rate of false positives. In addition, while deploying BOTMAGNIFIER in a production system, one could add a filter that applies the techniques from Section 3.6.1 to any magnified pool, and obtain clean results that he could use for spam reduction.

Improving existing blacklists. We wanted to understand whether our approach can improve existing blacklists by providing information about spamming bots that are currently active. To achieve this, we analyzed the email logs from the UCSB computer sci-

ence department over a period of two months, from November 30, 2010 to February 8, 2011. As a first step, the department mail server uses *Spamhaus* as a pre-filtering mechanism, and therefore the majority of the spam gets blocked before being processed. For each email whose sender is not blacklisted, the server runs *SpamAssassin* [29] for content analysis, to find out if the message content is suspicious. *SpamAssassin* assigns a spam score to each message, and the server flags it as spam or ham according to that value. These two steps are useful to evaluate how BOTMAGNIFIER performs, for the following reasons:

- If a mail reached the server during a certain day, it means that at that time its sender was not blacklisted by *Spamhaus*.
- The spam ratios computed by *SpamAssassin* provide a method for the evaluation of BOTMAGNIFIER's false positives.

During the analysis period, the department mail server logged 327,706 emails in total, sent by 228,297 distinct IP addresses. Of these, 28,563 emails were considered as spam by *SpamAssassin*, i.e., they bypassed the first filtering step based on *Spamhaus*. These emails had been sent by 10,284 IP addresses. We compared these IP addresses with the magnified sets obtained by BOTMAGNIFIER during the same period: 1,102 ($\approx 10.8\%$) appeared in the magnified sets. We then evaluated how many of these IP addresses would have been detected before reaching the server if our tool would have been used in parallel with the DNSBL system. To do this, we analyzed how many of the spam sender IP addresses were detected by BOTMAGNIFIER before they sent spam to our server. We found 295 IP addresses showing this behavior. All together, these hosts sent

1,225 emails, which accounted for 4% of the total spam received by the server during this time.

We then wanted to quantify the false positives in the magnified pools generated by BOTMAGNIFIER. To do this, we first searched for those IP addresses that were in one of the magnification pools, but had been considered sending ham by *SpamAssassin*. This resulted in 28 matches. Of these, 15 were blacklisted by *Spamhaus* when we ran the tests, and therefore we assume they are false negatives by *SpamAssassin*. Of the remaining 13 hosts, 12 were detected as legitimate servers by the filters described in Section 3.6.1. For the remaining one IP address, we found evidence of it being associated with spamming behavior on another blacklist [107]. We therefore consider it as a false negative by *SpamAssassin* as well.

In summary, we conclude that BOTMAGNIFIER can be used to improve the spam filtering on the department email server: the server would have been reached by 4% less spam emails, and no legitimate emails would have been dropped by mistake within these two months. Having access to more *Spamhaus* mirrors would allow us to increase this percentage.

Resilience to evasion. If the techniques introduced by BOTMAGNIFIER become popular, spammers will modify their behavior to evade detection. In this section, we discuss how we could react to such evasion attempts.

The first method that could be used against our system is obfuscating the email subject lines, to prevent BOTMAGNIFIER from creating the seed pools. If this was the case,

we could leverage previous work [104, 148] that takes into account the body of emails to identify emails that are sent by the same botnet. As an alternative, we could use different methods to build the seed pools, such as clustering bots based on the IPs of the C&C servers that they contact.

Another evasion approach that spammers might try is to reduce the number of bots associated with each campaign. The goal would be to stay under the threshold required by BOTMAGNIFIER (i.e., 1,000) to work. This would require more management effort on the botmaster's side, since more campaigns would need to be run. Moreover, we could use other techniques to cluster the spam campaigns. For example, it is unlikely that the spammers would set up a different website for each of the small campaigns they create. We could then cluster the campaigns by looking at the web sites the URLs in the spam emails point to.

Other evasion techniques might be to assign a single domain to each spamming bot, or to avoid evenly distributing email lists among bots. In the first case, BOTMAGNIFIER would not be able to unequivocally identify a bot as being part of a specific botnet. However, the attribution requirement could be dropped, and these bots would still be detected as generic spamming bots. The second case would be successful in evading our current system. However, this behavior involves something that spammers want to avoid: having the same bot sending thousands of emails to the same domain within a short amount of time would most likely result in the bot being quickly blacklisted.

3.6.4 Universality of k

In Section 3.4, we introduced a function to determine the optimal N value according to the size of the seed pool's target $|T(p_i)|$. To do this, we analyzed the data from two C&C servers of the *Cutwail* botnet. One could argue that this parameter will work well only for campaigns carried out by that botnet. To demonstrate that the value of k (and subsequently of N) estimated by the function produces good results for campaigns carried out by other botnets, we ran the same precision versus recall technique we used in Section 3.4 on other datasets. Specifically, we analyzed 600 campaigns observed in the wild, which had been carried out by the other botnets we studied (*Lethic*, *Rustock*, *Waledac*, and *MegaD*). Since we did not have access to full ground truth for these campaigns, we used the IP addresses from the seed pools as true positives, and the set of IP addresses not blacklisted by *Spamhaus* as false positives. For the purpose of this analysis, we ignored any other IP address returned by the magnification process (i.e., magnified IP addresses already blacklisted by *Spamhaus*).

The results are shown in Figure 3.7. The figure shows the function plot of k in relation to the size of $|T(p_i)|$. The dots show, for each campaign we analyzed, where the optimal value of k lies. As it can be seen, the function of k we used approximates the optimal values for most campaigns well. This technique for setting k might also be used to set up BOTMAGNIFIER in the wild, when ground truth is not available.

Data stream independence. In Section 3.2.2, we claimed that BOTMAGNIFIER can work with any kind of transaction log as long as this dataset provides information about which IP addresses sent email to which destination email servers at a given point in

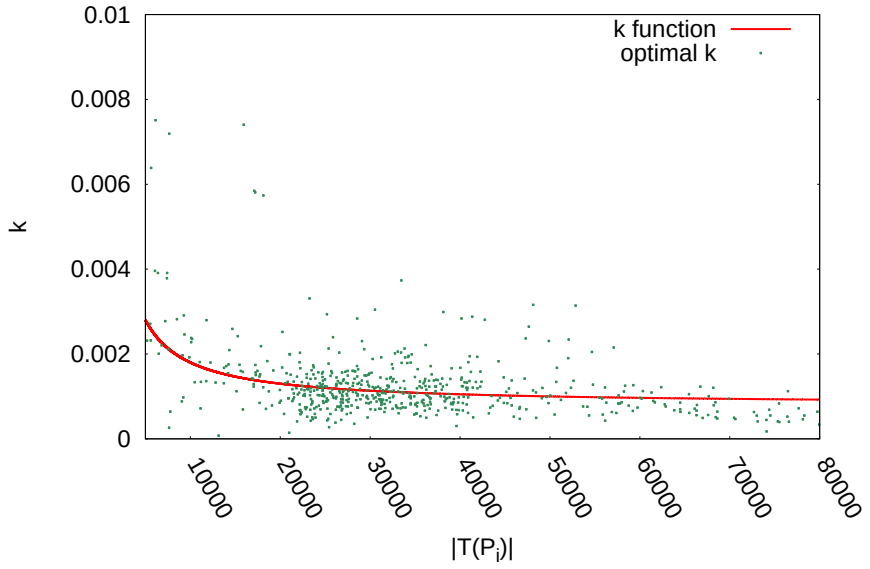


Figure 3.7: Analysis of our function for k compared to the optimal value of k for 600 campaigns.

time. To confirm this claim, we ran BOTMAGNIFIER on an alternative dataset, extracted from *netflow records* [50] collected by the routers of a large Internet service provider. The netflow data is collected with a sampling rate of 1 out of 1,000. To extract the data in a format BOTMAGNIFIER understands, we extracted each connection directed to port 25 TCP and considered the timestamp in which the connection initiated as the time the email was sent. On average, this transaction log contains 1.9 million entries per day related to about 194,000 unique sources.

To run BOTMAGNIFIER on this dataset, we first need to correctly dimension k. As explained in Section 3.4, the equation for k is stable for any transaction log. However, the value of the constants k_b and α changes for each dataset. To correctly dimension these parameters, we ran BOTMAGNIFIER on several campaigns extracted from the

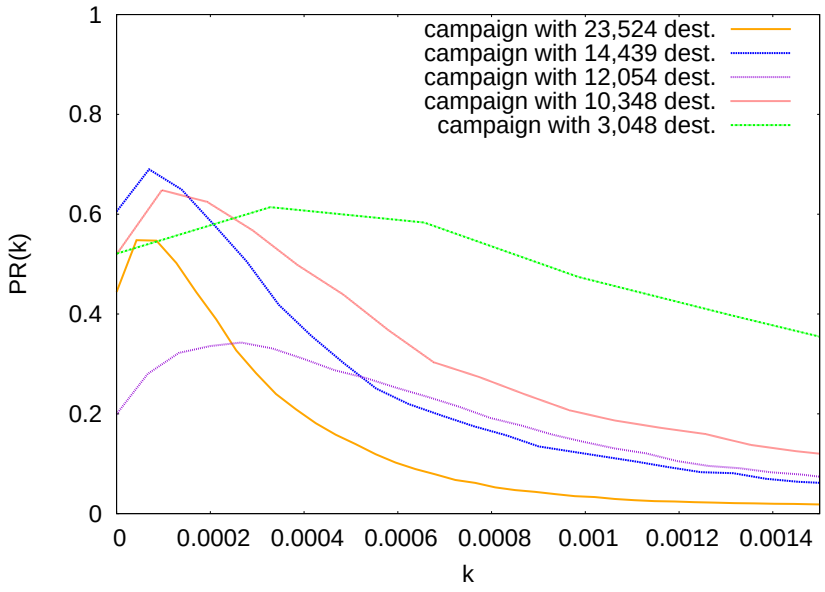


Figure 3.8: Precision vs. Recall functions for five campaigns observed in the netflow dataset.

netflow records. The $PR(k)$ analysis is shown in Figure 3.8. The optimal point of the campaigns is located at a lower k for this dataset compared to the ones analyzed in Section 3.4. To address this difference, we set k_b to 0.00008 and α to 1 when dealing with netflow records as transaction logs. After setting these parameters, we analyzed one week of data with BOTMAGNIFIER. The analysis period was between January 20 and January 28, 2011. During this period, we tracked 94,894 bots. Of these, 36,739 ($\approx$ 38.7%) belonged to the magnified sets of the observed campaigns. In particular, we observed 40,773 *Rustock* bots, 20,778 *Lethic* bots, 6,045 *Waledac* bots, and 1,793 *Cutwail* bots.

3.7 Conclusions

In this chapter we presented BOTMAGNIFIER, a tool for tracking and analyzing spamming botnets. The tool is able to "magnify" an initial seed pool of spamming IP addresses by learning the behavior of known spamming bots and matching the learned patterns against a (partial) log of the email transactions carried out on the Internet. We have validated and evaluated our approach on a number of datasets (including the ground truth data from a botnet's C&C hosts), showing that BOTMAGNIFIER is indeed able to accurately identify and track botnets. We showed BOTMAGNIFIER can be effectively used to reduce the amount of spam received by email servers, as well as to complement existing IP blacklists.

Part II

Detecting Misbehaving Accounts

Chapter 4

Background: Online Social Networks

Online social networks (OSNs) offer a way for users to keep track of their friends and communicate with them. To this end, a network of trust typically regulates which personal information is visible to whom. For example, users can establish friend relationships with each other, and start seeing the content that their friends post on the social network.

Online social networks witnessed an exponential growth over the last years. Early platforms such as *Livejournal* were created in the late nineties, and were fairly simple [8]. At the time of writing, there are many large social networks with hundreds of millions users each, and this count keeps growing. Even sites that were not born as social networks, such as *Youtube* [19], include "social" elements, with users who can connect with each other and comment on each other's content. In general, the boundary between social networks and regular websites is blurring. For example, news sites include on

their pages buttons to easily share or like content on social networks.

From a security point of view, social networks have unique characteristics. First, information access and interaction is based on trust. Users typically share a substantial amount of personal information with their friends. This information may be public or not. If it is not public, access to it is regulated by a network of trust. In this case, a user allows only her friends to view the information regarding herself. Unfortunately, social networking sites do not provide strong authentication mechanisms, and it is easy to impersonate a user and sneak into a person's network of trust [99]. Moreover, it often happens that users, to gain popularity, accept any friendship request they receive, exposing their personal information to people they do not know in real life. In other cases, such as MySpace, the information displayed on a user's page is public by design. Therefore, anyone can access it, friend or not. Networks of trust are important from a security point of view, because they are often the only mechanism that protects users from being contacted by unwanted entities.

Another distinguishing characteristic of social networks is that Internet users present a different awareness to threats on social networks than on other types of online services. While most users have become aware of the common threats that affect the Internet, such as e-mail spam and phishing, they usually do not show an adequate understanding of the threats hidden in social networks. For example, a previous study showed that 45% of users on a social networking site readily click on links posted by their "friend" accounts, even if they do not know that person in real life [35]. This behavior might be abused by spammers who want to advertise web sites, and might be particularly harmful to users if spam messages contain links to web pages serving malware.

As part of my research I develop systems to detect and block malicious activity on online social networks. In this thesis I focused on malicious activity carried on three social networks: Facebook, Twitter, and MySpace. The reason of this choice is that these social networks were the largest ones when this research work started. Facebook and Twitter are still the largest social networks, while MySpace witnessed a steady decline in users. Although our studies are limited to three social networks, other social networking sites present similar characteristics, and therefore our detection approaches can be adapted to them.

In the remainder of this chapter I briefly describe the social networks that we analyzed in our work.

4.1 The Facebook Social Network

Facebook is currently the largest social network on the Internet. Measurement studies reported that the social network had 721 million million active users in 2011 [141], with over 2 billion media items (videos and pictures) shared every week [4].

User profiles are not public by default, and the right to view a user's page is granted only after having established a relationship of trust (paraphrasing the Facebook terminology, *becoming friends*) with the user. When a user A wants to become friend with another user B, the platform first sends a request to B, who has to acknowledge that she knows A. When B confirms the request, a friendship connection with A is established. Previous research showed that the perception of Facebook friendship by users

is different from their perception of a relationship in real life [35]. Most of the time, Facebook users accept friendship requests from people they barely know, while in real life a person would be stricter on who she considers a "friend."

When they visit the Facebook website, users are presented with a timeline, which contains the updates recently shared by the user's friends on the social network. A user can separate their friends in groups, and specify what type of content is visible to each group.

In the past, most Facebook users were grouped in *networks*, where people coming from a certain country, town, or school could find their neighbors or peers. The default privacy setting for Facebook was to allow all people in the same network to view each other's profiles. Thus, a malicious user could join a large network to crawl data from the users on that network. Among the rest, this data allowed an adversary to carry out targeted attacks. For example, a spammer could run a campaign that targets only those users whose profiles have certain characteristics (e.g., gender, age, interests) and who, therefore, might be more responsive to that campaign. For this reason, Facebook deprecated geographic networks in October 2009. School and company networks are still available, but their security is better, since to join one of these networks, a user has to provide a valid e-mail address from that institution (e.g., a university e-mail address).

4.2 The MySpace Social Network

MySpace used to be the largest social network on the Internet. Over the last years, the side steadily lost users, who mostly moved to Facebook [2]. In 2013 the site witnessed a major restyling, and it became mostly focused on sharing music and videos.

When we performed the studies described in this thesis, in 2009, MySpace was still a "traditional" online social network. The basic idea of this network was to provide each user with a web page, which the user was able to personalize with information about herself and her interests. Even though MySpace also had the concept of "friendship," like Facebook, MySpace pages were public by default. Therefore, it was easier for a malicious user to obtain sensitive information about a user on MySpace than on Facebook. It was possible to profile users by gender, age, or nationality, and an aimed spam campaign could target a specific group of users to enhance its effectiveness.

4.3 The Twitter Social Network

Twitter is a much simpler social network than Facebook and MySpace. It is designed as a microblogging platform, where users send short text messages (i.e., *tweets*) that appear on their friends' feeds (similar to Facebook's timeline).

Unlike Facebook and MySpace, no personal information is shown on Twitter pages by default. Users are identified only by a *profile name* and, optionally, by a real name. To profile a user, it is possible to analyze the tweets she sends, and the feeds to which

she is subscribed. However, this is significantly more difficult than on the other social networks.

A Twitter user can start "following" another user. As a consequence, she starts receiving the user's tweets on her own page. The user who is "followed" can, if she wants, follow the other one back. This makes Twitter different from other social networks, because the "friend" relation does not have to be mutual, but actually it is often unilateral.

Tweets can be grouped by *hashtags*, which are popular words, beginning with a "#" character. This allows users to efficiently search who is posting topics of interest at a certain time. When a user likes someone's tweet, she can decide to *retweet* it. As a result, that message is shown to all her followers. By default, profiles on Twitter are public, but a user can decide to protect her profile. If she chooses to do so, the user will have to authorize people before they can follow her.

Although it is usually considered a social network, the asymetry of user relations on Twitter makes this service very different from other OSN sites. Previous research showed that many users consider this network as a news platform instead [86].

Twitter witnessed a constant growth over the last years. At the time of writing, Twitter is used by more than 240 million active users every month.

In the next chapter I present our approach to detecting fake accounts that are misused by cybercriminals on online social networks. Then, in Chapter 6, I present a system able to detect that a legitimate online social network account has been compromised and is now misused by a cybercriminal.

Chapter 5

Detecting Fake Online Social Network Accounts

5.1 Introduction

As any popular online service, social networks have to deal with misuse too. One way that miscreants misuse social networks is by creating fake accounts and using them to spread spam or other malicious content on the site [63]. In this chapter, we present the results of a year-long study of spam activity on social networks. By leveraging these observations, we developed SPAMDETECTOR, a system to automatically flag fake accounts that are misused by cybercriminals. SPAMDETECTOR was one of the first tools developed in the research community to detect malicious accounts on social networks.

In this chapter we make the following contributions:

- We created a set of honeypot accounts (honey-profiles) on three major social networks, and we logged all the activity (malicious or not) that these accounts were able to observe over a one-year period for Facebook and an eleven-month period for Twitter and MySpace.

- We investigate how spammers are using social networks, and we examine the effectiveness of the countermeasures that are taken by the major social network portals to prevent spamming on their platforms.

- We identify characteristics that allow us to detect fake accounts controlled by spammers on a social network.

- We built SPAMDETECTOR, a tool to detect spammers, and used it on a Twitter and Facebook dataset. We obtained some promising results. In particular, we correctly detected 15,857 on Twitter and, after our submission to the Twitter spam team, these accounts were suspended.

This chapter is structured as follows: In Section 5.2, I describe the setup of our experiment, focusing on the parameters we used to create the honey-profiles, as well as the kind of data that we collected. In Section 5.3, I analyze the collected data and describe our findings about spamming activity. In Section 5.4, I introduce SPAMDETECTOR and the techniques for the detection of spam profiles on social networks that it uses.

5.2 Data Collection

Our first goal is to understand the extent to which spam is a problem on social networks, as well as the characterization of accounts used to spread spam. To this end, we created 900 profiles on Facebook, MySpace, and Twitter, 300 on each platform. The purpose of these accounts was to log the traffic (e.g., friend requests, messages, invitations) they received from other users of the network. Due to the similarity of these profiles to honeypots [7], we call these accounts *honey-profiles*.

5.2.1 Honey-Profiles

Our goal was to create a number of honey-profiles that reflect a representative selection of the population of the social networks we analyzed. To this end, we first crawled each social network to collect common profile data.

On Facebook, we joined 16 geographic networks, using a small number of manually-created accounts. This was possible because, at the time, geographic networks were still available; as we mentioned in Chapter 4, this feature was discontinued in 2009.

Since we wanted to create profiles reflecting a diverse population, we joined networks on all continents (except Antarctica and Oceania): the Los Angeles and New York networks for North America, the London, France, Italy, Germany, Russia, and Spain ones for Europe, the China, Japan, India, and Saudi Arabia ones for Asia, the Algeria and Nigeria ones for Africa, and the Brazil and Argentina networks for South America. For each network, we crawled 2,000 accounts at random, logging names, ages,

and gender (which is the basic information required to create a profile on Facebook). Afterwards, we randomly mixed this data (names, surnames, and ages) and created the honey-profiles. Gender was determined by the first name. Each profile was assigned to a network. Accounts created using data from a certain network were assigned either to this network or to a network where the main language spoken was the same (e.g., profiles created from accounts in the France network were used in networks associated with francophone countries). This was a manual process. For larger networks (e.g., New York, Germany, Italy) up to three accounts were created, while only one account was set up for smaller ones. In total, we created 300 accounts on the Facebook platform.

On MySpace, we crawled 4,000 accounts in total. This was easier than on Facebook because, as mentioned in Chapter 4, most profile pages are public. Similar to Facebook, our aim was to generate "average" profiles based on the user population of the social network. After data collection, we looked for common names and ages from profiles with different languages, and created profiles in most nations of the world. We created 300 accounts on MySpace for our experiment.

On Facebook and MySpace birth date and gender are needed for registration. On Twitter, on the other hand, the only information required for signing up is a full name and a profile name. Therefore, we did not find it necessary to crawl the social network for "average" profile information, and we simply used first names and surnames from the other social networks. For each account, the profile name has been chosen as a concatenation of the first and last name, plus a random number to avoid conflicts with already existing accounts. Similarly to the other networks, we created 300 profiles.

We did not create more than 300 profiles on each network because registration is a semi-

automated process. More precisely, even though we could automatically fill the forms required for registration, we still needed a human to solve the CAPTCHAs involved in the process.

5.2.2 Collection of Data

After having created our honey-profiles, we ran scripts that periodically connected to those accounts and checked for activity. We decided that our accounts should act in a passive way. Therefore, we did not send any friend requests, but accepted all the ones that were received.

In a social network, the first action a malicious user would likely execute to get in touch with his victims is to send them a friend request. This might be done to attract the user to the spammer's profile to view the spam messages (on MySpace), lure the victim into following back the fake accounts (on Twitter), or to invite her to accept the friendship and start seeing the spammer's messages in her own feed (on Facebook).

After having acknowledged a request (i.e., accepted the friendship on Facebook and MySpace or started following the user on Twitter), we logged all the information needed to detect malicious activity. More precisely, we logged every email notification received from the social networks, as well as all the requests and messages seen on the honey-profiles. On some networks, such as Facebook, the notifications and messages might be of different types (e.g., application and group invitations, video posts, status messages, private messages), while on other platforms, they are more uniform (e.g., on Twitter, they are always short text messages). We logged all types of requests on Facebook, as

well as wall posts, status updates, and private messages. On MySpace, we recorded mood updates, wall posts, and messages. On Twitter, we logged tweets and direct messages.

Our scripts ran continuously for 12 months for Facebook (from June 6, 2009 to June 6, 2010), and for 11 months for MySpace and Twitter (from June 24, 2009 to June 6, 2010), periodically visiting each account. The visits had to be performed slowly (approximately one account visited every 2 minutes) to avoid being detected as a bot by the social networking site and, therefore, having the accounts deleted.

5.3 Analysis of Collected Data

As mentioned previously, the first action that a spammer would likely execute is to send friend requests to her victims. Only a fraction of the contacted users will acknowledge a request, since they do not know the real-life person associated with the account used by the bot[1]. On Twitter, the concept of friendship is slightly different, but the *modus operandi* of the spammers controlling fake accounts is the same: they start following victims, hoping that they will follow them back, starting to receive the spam content. From the perspective of our analysis, friendships and mutual follow relationships are equivalent. When a user accepts one of the friend requests, she lets the spammer enter her network of trust. In practice, this action has a major consequence: The victim starts to see messages received from the spammer in her own news/message feed. This kind of spamming is very effective, because the spammer has to write a single message

[1]We assume that most spam accounts are managed in an automated fashion. Therefore, from this point on, we will use the terms fake account and *bots* interchangeably.

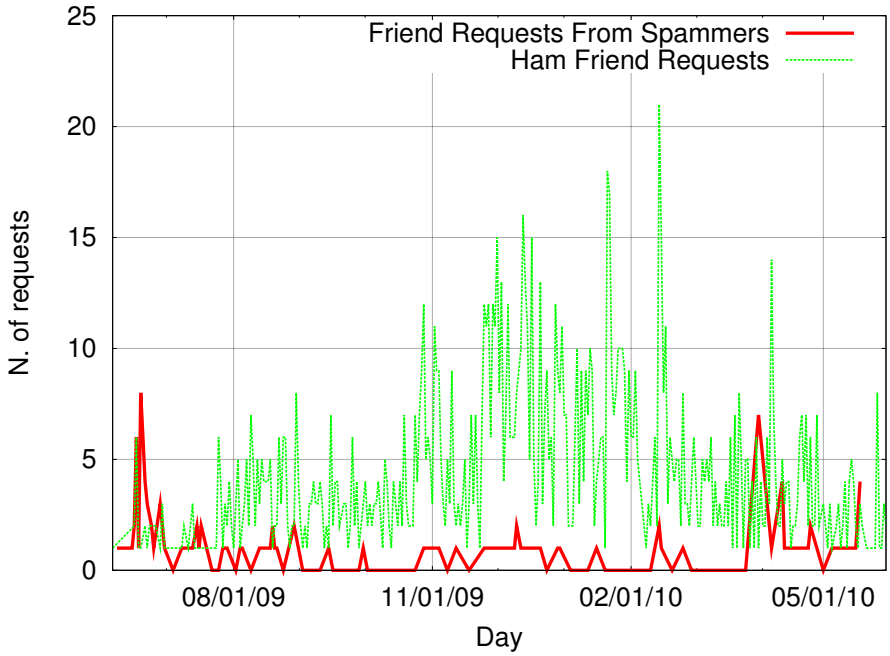


Figure 5.1: Friend requests received by our honey-profiles on Facebook.

(e.g., a status update on Facebook), and the message appears in the feeds of all the victims. Depending on the social network, the nature of these messages can change: they are *status updates* on Facebook, *status* or *mood updates* on MySpace, and *tweets* on Twitter.

During our study, our honey-profiles received a total of 4,250 friend requests. As can be seen in Table 5.1, the amount of requests varies from network to network. This

Network	**Overall**	**Spammers**
Facebook	3,831	173
MySpace	22	8
Twitter	397	361

Table 5.1: Friend requests received by our honey-profiles on the various social networks.

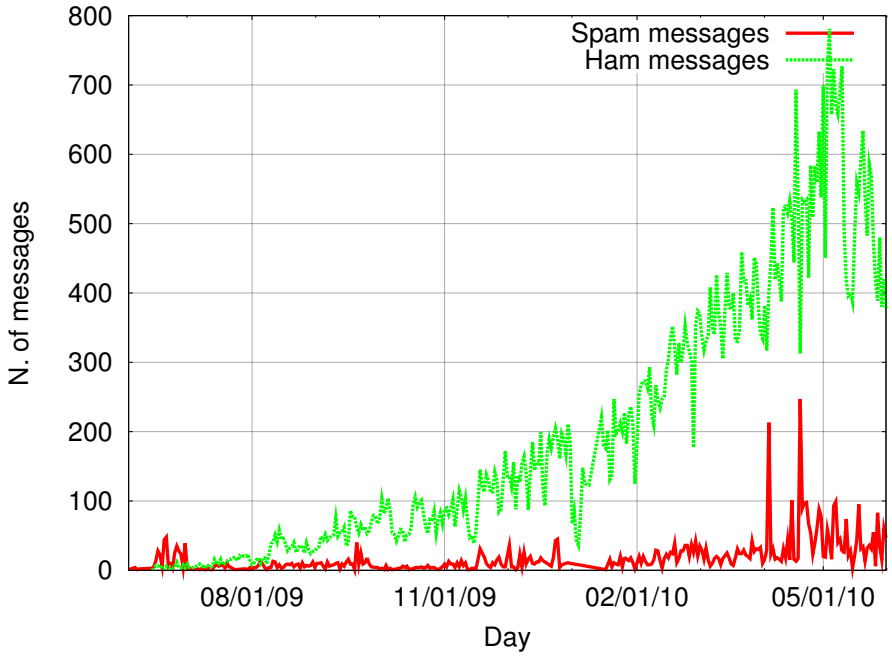


Figure 5.2: Messages observed by our honey-profiles on Facebook.

might be caused by the different characteristics of the various social networks. As one would expect, we observed the largest amount of requests on Facebook, since it has the largest user base. Surprisingly, however, the majority of these requests proved not to come from spam bots, but from real users, looking for popularity or for real people with the same name as one of our honey-profiles. Another surprising finding is that, on MySpace, we received a very low number of friend requests. It is not clear what the

Network	**Overall**	**Spammers**
Facebook	72,431	3,882
MySpace	25	0
Twitter	13,113	11,338

Table 5.2: Messages received by our honey-profiles on the various social networks.

reason of the disparity between this social network and Facebook is, since MySpace also provided a mechanism to easily post messages on users' pages. Daily statistics for friend requests received by our honey-profiles on Facebook and Twitter are shown in Figures 5.1 and 5.3.

Information about the logged messages is shown in Table 5.2. Overall, we observed 85,569 messages. Again, there is a big disparity between the three social networks. On Twitter, interestingly, we recorded the largest amount of spam messages. Given the smaller size of the network's user base, this is surprising. Daily statistics for messages received on Facebook are shown in Figure 5.2, while those for Twitter are reported in Figure 5.4. We do not show a graph for MySpace because the number of messages we received was very low.

On Facebook, we also observed a fair amount of invitations to applications, groups, and events, as well as posting of photos and videos in our honey-profiles' feeds. However, since none of them were spam, but were coming from legitimate accounts that mistakenly connected to our honey-profiles, we ignored them for the subsequent analysis.

5.3.1 Identification of Spam Accounts

Tables 5.1 and 5.2 show the breakdown of requests that were received by our honey-profiles. We can see that the honey-profiles did not only receive friend requests and messages from spammers, but also a surprising amount from legitimate accounts. Even if friend requests are unsolicited, they are not always the result of spammers who reach out. In particular, many social network users aim to increase their popularity by adding

as friends people they do not know. On Facebook, since all our honey-profiles were members of a geographic network (as long as these were available), it is also possible that people looking for local "friends" would have contacted some of our accounts. In particular, we observed that this occurs with more frequency on smaller networks (in particular, some Middle Eastern and African ones). Moreover, since we picked random combinations of first and last names, it happened that some of our honey-profiles had the same name as a real person, and, as a consequence, the account was contacted by real friends of this person. Since not all friend requests and messages are malicious, we had to distinguish between spammers and benign users.

To discriminate between real users and spam bots, we started to manually check all the profiles that contacted us. During this process, we noticed that fake accounts used to spread spam share some common traits, and formalized them in features that we then used for automated spam detection. We will describe these features in detail in Section 5.4.

We found that, of the original 3,831 accounts that contacted us on Facebook, 173 were spammers. Moreover, on Facebook, during the last months of logging, the ratio of spam messages compared to legitimate ones dramatically dropped. The reason is that when a legitimate user adds our honey-profile to her friend list, this honey-profile starts appearing on her friends' pages as a friend suggestion. This leads to a number of additional friend requests (and messages) from real users. On MySpace, we detected 8 spammers. On Twitter, we detected 361 spammers out of 397 accounts that contacted our honey-profiles.

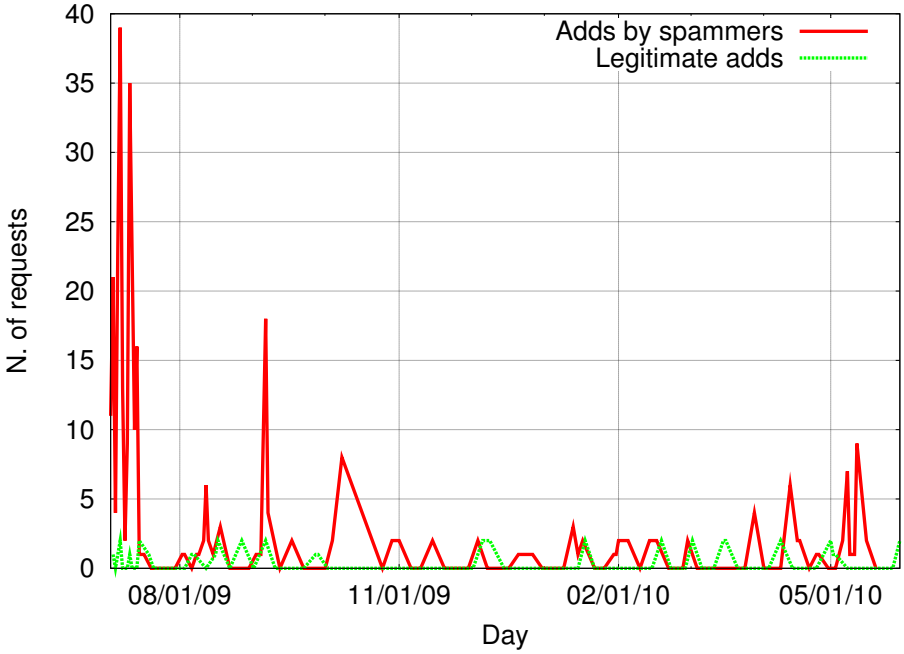


Figure 5.3: Users starting following our honey-profiles on Twitter.

5.3.2 Spam Bot Analysis

The fake accounts that we identified showed different levels of activity and different strategies to deliver spam. Based on their spam strategy, we distinguish four categories of bots:

1. **Displayer**: Bots that do not post spam messages, but only display some spam content on their own profile pages. In order to view spam content, a victim has to manually visit the profile page of the bot. This kind of bots is likely to be the least effective in terms of people reached. All the detected MySpace bots belonged to this category, as well as two Facebook bots.

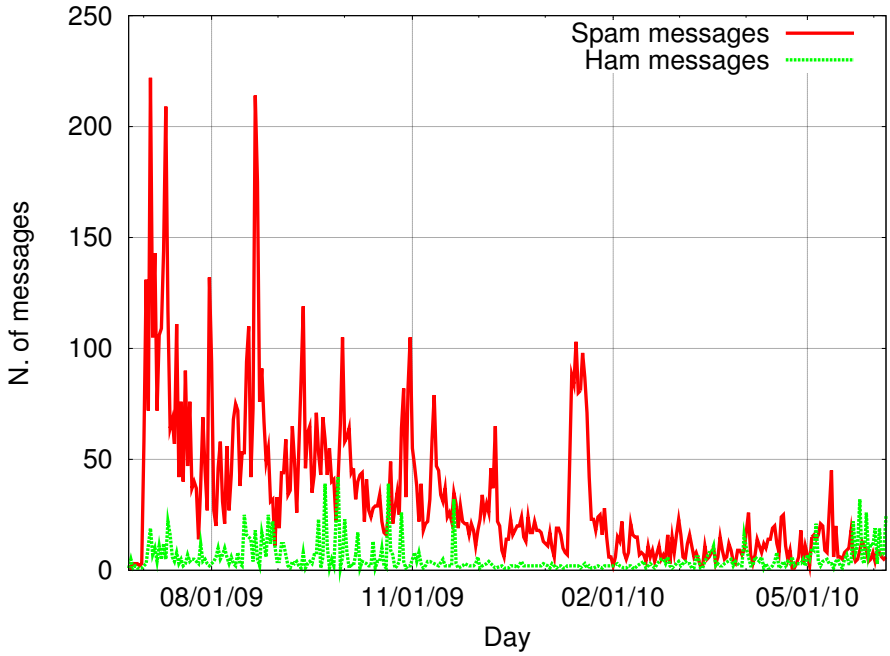


Figure 5.4: Messages received by our honey-profiles on Twitter.

2. **Bragger**: Bots that post messages to their own feed. These messages vary according to the networks: on Facebook, these messages are usually status updates, while on Twitter these are tweets. The result of this action is that the spam message is distributed and shown on all the victims' feeds. However, the spam is not shown on the victim's profile when the page is visited by someone else (i.e., one of the victim's friends). Therefore, the spam campaign reaches only victims who are directly connected with the spam bot. 163 bots on Facebook belonged to this category, as well as 341 bots on Twitter.

3. **Poster**: Bots that send a direct message to each victim. This can be achieved in different ways, depending on the social network. On Facebook, for example, the message might be a post on a victim's wall. The spam is shown on the victims

feed, but, unlike the case of a "bragger," can be viewed also by the victim's friends visiting her profile page. This is the most effective way of spamming, because it reaches a greater number of users compared to the previous two. Eight bots from this category have been detected, all of them on the Facebook network. Koobface-related messages also belong to this category (see [30]).

4. **Whisperer**: Bots that send private messages to their victims. As for "poster" bots, these messages have to be addressed to a specific user. The difference, however, is that this time the victim is the only one seeing the spam message. This type of bots is fairly common on Twitter, where spam bots send direct messages to their victim. We observed 20 bots of this kind on this network, but none on Facebook and MySpace.

We then examined the message-sending activity of spam bots on different networks. On Facebook, we observed an average of 11 spam messages per day, while, on Twitter, the average number of messages observed was 34. On MySpace, we did not observe any spam message. The reason is that all the spam bots on MySpace are "displayers." The difference between Twitter and Facebook activity is caused by the apparently different responses of the two social networks to spam. More precisely, we observed that, at the time of this study, Facebook seemed to be much more aggressive in fighting spam. This is demonstrated by the fact that, on Facebook, the average lifespan of a spam account was four days. On Twitter, on the other hand, this lifespan it was 31 days. No spam accounts were deleted during our observation on MySpace.

As shown in Figures 5.1 and 5.3, many spam requests arrived during the first days of our experiment, especially on Facebook. All the early-days spammers have been quickly

deleted by Facebook (the one with the longest life lasted one month), while most of the Twitter ones were deleted only after we reported them to their spam team.

It is also interesting to look at the time of the day when messages and friend requests are sent. The reason is that fake accounts might be accessed periodically or at specific times to send their messages. Benign activity, on the other hand, follows the natural diurnal pattern. During our observation, we noticed that some bots showed a higher activity around midnight (GMT -7), while in the same period of time, the legitimate messages registered a low.

Another way to study the effectiveness of spam activity is to look at how many users acknowledged friend requests on the different networks. On Facebook, the average number of confirmed friends of spam bots is 21, on MySpace it is 31, while on Twitter it is 350. We assume that the difference in number of people reached is probably due to the different lifetime of the bots in the different networks. The low activity of the bots on MySpace might be the cause of both the low numbers of bots detected on that network and their longer lifetime.

We identified two kinds of bot behavior: stealthy and greedy bots. Greedy ones include spam content in every message they send. They are easier to detect, and might lead users to flag bots as spammers or to revoke their friendship status. Stealthy bots, on the other hand, send messages that look legitimate, and only once in a while inject a message containing spam content. Since they look like legitimate profiles, they might convince more people to accept and maintain friendships.

Of the 534 spam bots detected, 416 were greedy and 98 were stealthy (note that ten spam profiles were "displayers," and 20 were "whisperers." These bots, therefore, did not use status updates or tweets to spam).

Another interesting observation is that spam bots are usually less active than legitimate users. This probably happens because sending out too many messages would make detection by the social network too easy. For this reason, most spam profiles we observed, both on Facebook and Twitter, sent less than 20 messages during their life span.

While observing Facebook spammers, we also noticed that many of them did not seem to pick victims randomly, but, instead, they seemed to follow certain criteria. In particular, most of their victims happened to be male. This was particularly true for campaigns advertising adult websites. Since Facebook does not provide an easy way to search for people based on gender, the only way spammers can identify their victims is by looking for male first names. This intuition led us to another observation. The list of victims targeted by these bots usually shows an anomalous repetition of people with the same first name (e.g., tens of profiles with only four different given names). This might happen because spam bots are given lists of common first names to target. In addition, Facebook people search does not make a difference between first and last name while searching. For this reason, these gender-aware bots sometimes targeted female users who happened to have a male name as last name (e.g., Wayne).

Mobile Interface Most social networking sites have introduced techniques to prevent automatic account generation and message sending. For example, at the time of this study a Facebook user was required to solve a CAPTCHA [12] every time she tries to

send a friend request. A CAPTCHA has to be solved also every time an account is created. Moreover, the site uses a very complicated JavaScript environment that makes it difficult for bots to interact with the pages. On the other hand, the complexity of these sites made them not very attractive to mobile Internet users, who use less powerful devices and slower connections.

To attract more users and to make their platform more accessible from any kind of device, major social networks launched mobile versions of their sites. These versions offer the main functionality of the complete social networking sites, but in a simpler fashion. To improve usability, no JavaScript is present on these pages, and no CAPTCHAs are required to send friend requests. This has made social networks more accessible from everywhere. However, the mobile environment provides spammers with an easy way to interact with these sites and carry out their tasks. This is confirmed by our analysis: 80% of the bots we detected on Facebook used the mobile site to send their spam messages. However, to create a fake account, it is still necessary to go through the non-mobile version of the site. For Twitter spam, there is no need for the bots to use the mobile site, since an API to interact with the network is provided, and, in any case, there is no need to solve CAPTCHAs other than the one needed to create a profile.

5.4 Spam Profile Detection

Based on our understanding of spam activity on social networks, the next goal was to leverage these insights to develop techniques to detect spammers in the wild. To this end, we developed a system, called SPAMDETECTOR. We decided to focus on

detecting "bragger" and "poster" spammers, since they do not require real profiles for detection, but are just detectable by looking at their own feeds. We used machine learning techniques to classify spammers and legitimate users. To detect whether a given profile belongs to a spammer or not, we developed six features, which are:

FF ratio (R): The first feature compares the number of friend requests that a user sent to the number of friends she has. Since a bot is not a real person, and, therefore, nobody knows him/her in real life, only a fraction of the profiles contacted would acknowledge a friend request. Thus, one would expect a distinct difference between the number of friend requests sent and the number of those that are acknowledged. More precisely, we expect the ratio of friend requests to actual friends to be large for spammers and low for regular users. Unfortunately, the number of friend requests sent is not public on Facebook and on MySpace. On Twitter, on the other hand, the number of users a profile started to follow is public. Therefore, we can compute the ratio $R = \textit{following} \ / \ \textit{followers}$ (where following, in the Twitter jargon, is the number of friend requests sent, and followers is the number of users who accepted the request).

URL ratio (U): The second feature to detect a bot is the presence of URLs in the logged messages. To attract users to spam web pages, bots are likely to send URLs in their messages. Therefore, we introduce the ratio U as:

$$U = \textit{messages\_containing\_urls} \ / \ \textit{total\_messages}.$$

Since, in the case of Facebook, most messages with URLs (link and video share, group invitations) contain a URL to other Facebook pages, we only count URLs pointing to a third party site when computing this feature.

Message Similarity (S): The third feature consists in leveraging the similarity among the messages sent by a user. Most bots we observed sent very similar messages, considering both message size and content, as well as the advertised sites. Of course, on Twitter, where the maximum size of the messages is 140 characters, message similarity is less significant than on Facebook and MySpace, where we logged messages up to 1,100 characters. We introduced the similarity parameter S, which is defined as follows:

$$S = \frac{\sum_{p \in P} c(p)}{l_a l_p},$$

where P is the set of possible message-to-message combinations among any two messages logged for a certain account, p is a single pair, $c(p)$ is a function calculating the number of words two messages share, l_a is the average length of messages posted by that user, and l_p is the number of message combinations. The idea behind this formula is that a profile sending similar messages will have a low value of S.

Friend Choice (F): The fourth feature attempts to detect whether a profile likely used a list of names to pick its friends or not. We call this feature F, and we define it as:

$$F = \frac{T_n}{D_n},$$

where T_n is the total number of names among the profiles' friend, and D_n is the number of distinct first names. Our observation showed that legitimate profiles have values of this feature that are close to 1, while spammers might reach values of 2 or more.

Messages Sent (M): We use the number of messages sent by a profile as a feature. This is based on the observation that profiles that sent out hundreds of messages are less

likely to be spammers, given that, in our initial analysis, most spam bots sent less that 20 messages.

Friend Number (FN): Finally we look at the number of friends a profile has. The idea is that profiles with thousands of friends are less likely to be spammers than the ones with a few.

Given our general set of features, we built two systems to detect spam bots on Facebook and Twitter. Since there are differences between these two social networks, some features had to be slightly modified to fit the characteristics of the particular social network. However, the general approach remains the same. We used the Weka framework [17] with a Random Forest algorithm [37] for our classifier. We chose this algorithm because it was the one that gave the best accuracy and lowest false positive ratio when we performed a ten-fold cross-validation on the training set.

5.4.1 Spam Detection on Facebook

The main issue when analyzing Facebook is to obtain a suitable amount of data to analyze. Most profiles are private, and only their friends can see their walls. At the beginning of this study, geographic networks were still available, but they were discontinued in October 2009. Therefore, we used data from various geographic networks, crawled between April 28 and July 8 2009, to test our approach. This dataset was described in detail in [146].

Since on Facebook the number of friend requests sent out is not public, we could not

apply the R feature.

We trained our classifier using 1,000 profiles. We used the 173 spam bots that contacted our honey-profiles as samples for spammers, and 827 manually checked profiles from the Los Angeles network as samples for legitimate users. A 10-fold cross validation on this training data set yielded an estimated false positive ratio of 2% and a false negative ratio of 1%. We then ran SPAMDETECTOR on 790,951 profiles, belonging to the Los Angeles and New York networks. We detected 130 spammers in this dataset. Among these, 7 were false positives. The reason for this low number of detected spammers might be that spam bots typically do not join geographic networks. This hypothesis is corroborated by the fact that among the spam profiles that contacted out honey profiles, none was a member of a geographic network. We then randomly picked 100 profiles, classified as legitimate. We manually looked at them looking for false negatives. None of them turned out to be a spammer.

5.4.2 Spam Detection on Twitter

On Twitter, it is much easier to obtain data than on Facebook, since most profiles are public. This gave us the possibility to deploy SPAMDETECTOR in the wild, and detect spammers in real time. The results of our analysis were then sent to Twitter, who verified that the accounts were indeed sending spam and removed them.

To train our classifier, we picked 500 spam profiles, coming either from the ones that contacted our honey profiles, or manually selected from the public timeline. We included profiles from the public timeline to increase diversity among spam profiles in

our training dataset. Among the profiles from the public timeline we chose the ones that stood out from the average for at least one of the R, U, and S features. We also picked 500 legitimate profiles from the public timeline. This was a manual process, to make sure that no spammers were miscategorized in the training set. The R feature was modified to reflect the number of followers a profile has. This was done because legitimate profiles with a fairly high number of followers (e.g., 300), but following thousands of other profiles, have a high value of R. This is a typical situation for legitimate accounts following news profiles, and would have led to false positives in our system. Therefore, we defined a new feature R', which is the R value divided by the number of followers a profile has. We used it instead of R for our classification.

After having trained the classifier, it was clear that the F feature is not useful to detect spammers on Twitter, since both spammers and legitimate profiles in the training set had very similar values for this parameter. This suggests that Twitter spam bots do not pick their victims based on their name. Therefore, we removed the F feature from the Twitter spam classifier. A 10-fold cross validation for the classifier with the updated feature set yielded an estimated false positive ratio of 2.5% and a false negative ratio of 3% on the training set.

Given the promising results on the training set and the possibility to access most profiles, we decided to use SPAMDETECTOR to detect spammers in real time on Twitter. The main problem that we faced while building our system was the crawling speed. Twitter limited our machine to execute only 20,000 API calls per hour. Thus, to avoid wasting our limited API calls, we performed Google searches for the most common words in tweets sent by the already detected spammers, and we crawled those profiles

that were returned as results. This approach has the problem that we can only detect profiles that send tweets similar to those of previously observed bots. To address this limitation, we created a public service where Twitter users can flag profiles as spammers. After a user flagged an account as a spammer, we ran our classifier on this profile data. If the profile is detected as a spammer, we add this profile to our detected spam set, enabling our system to find other profiles that sent out similar tweets.

Every time we detected a spam profile, we submitted it to Twitter's anti-spam team. During a period of three months, from March 06, 2010 to June 06, 2010, we crawled 135,834 profiles, detecting 15,932 of those as spammers. We sent this list of profiles to Twitter, and only 75 were reported by them to be false positives. All the other submitted profiles were suspended. In order to evaluate the false negative ratio, we randomly picked 100 profiles, classified as legitimate by our system. We then manually looked at them, finding out that 6 were false negatives.

To show that our targeted crawling did not affect our accuracy or false positive ratio, but just narrowed down the set of profiles to crawl, we picked 40,000 profiles at random from the public timeline and crawled them. Among these, we detected 102 spammers, with a single false positive. We can see that our crawling is effective, since the percentage of spammers in our targeted (crawled) dataset is 11%, whereas in the random set it is 0.25%. On the other hand, the false positive ratio on in both datasets is similarly low.

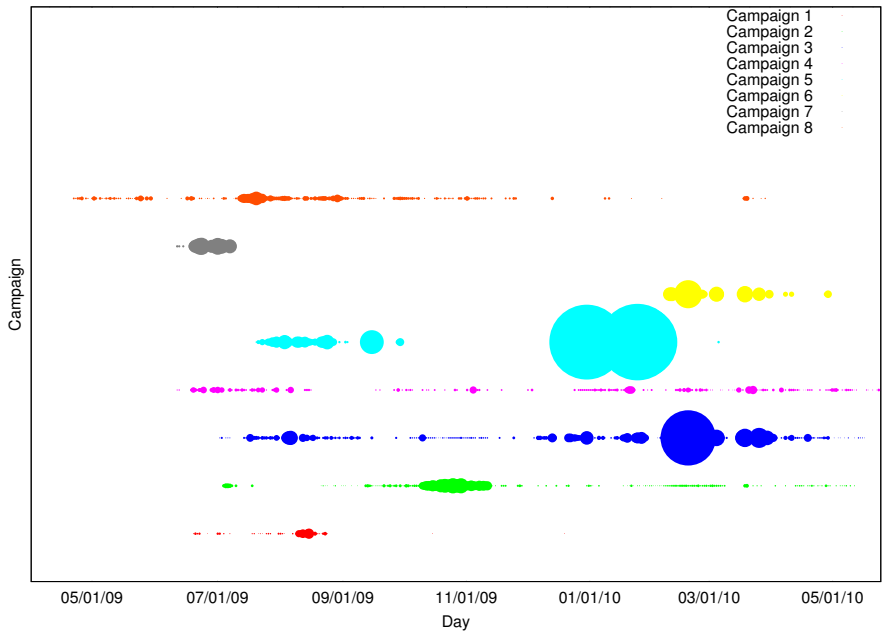


Figure 5.5: Activity of campaigns over time.

5.4.3 Identification of Spam Campaigns

After having identified single accounts used to send spam with SPAMDETECTOR, we analyzed the data to identify large-scale spam campaigns. With "spam campaign," we refer to multiple spam profiles that act under the coordination of a single spammer.

We consider two bots posting messages with URLs pointing to the same site as being part of the same campaign. Most bots hide the real URL that their links are pointing to by using URL shortening services (for example, *tinyurl* [15]). This is typically done to avoid easy detection by social network administrators and by the users, as well as to meet the message length requirements of some platforms (in particular, Twitter). To determine the actual site that a shortened URL points to, we visited all the URLs that

#	SN	Bots	# Mes.	Mes./day	Avg. vic.	Avg. lif.	G_c	Site adv.
1	T	485	1,020	0.79	52	25	0.28	Adult Dating
2	T	282	9,343	0.08	94	135	0.60	Ad Network
3	T,F	2,430	28,607	0.32	36	52	0.42	Adult Dating
4	T	137	3,213	0.15	87	120	0.56	Making Money
5	T,F	5,530	83,550	1.88	18	8	0.16	Adult Site
6	T,F	687	7,298	1.67	23	10	0.18	Adult Dating
7	T	860	4,929	0.05	112	198	0.88	Making Money
8	T	103	5,448	0.4	43	33	0.37	Ad Network

Table 5.3: Spam campaigns observed.

we observed. Then, we clustered all the profiles that advertised the same page. We list the top eight campaigns, based on the number of observed messages, in Table 5.3. Since we had most detections on Twitter, these campaigns targeted that network. It is interesting to notice, however, that bots belonging to three of them were observed on Facebook as well.

Some campaigns showed a large number of bots, each sending a few messages per day, while others send many messages using few bots. In addition, the fact that bots of a campaign can act in a stealthy or greedy way (see Section 5.3.2) leads to significantly different outcomes. Greedy bots that send spam with each message are easier to detect by the social network administrators. On the other hand, a low-traffic spam campaign is more difficult to detect. For example, the bots from Campaign 1 sent 0.79 messages per day, while the bots from the second campaign sent 0.08 messages per day on average. The result was that the bots from Campaign 1 have an average lifetime of 25 days, while the bots of Campaign 2 lasted 135 days on average. In addition, Campaign 2 reached more victims, as shown by an average of 94 friends (victims) per bot, while Campaign 1 only reached 52. This suggests that a relationship exists between the lifetime of bots

and the number of victims targeted. Clearly, an effective campaign should be able to reach many users, and having bots that live longer might be a good way to achieve this objective.

From the point of view of victims reached, stealthy campaigns are more effective. Campaigns 4 and 7 both used a stealthy approach. Of the messages sent, only 20-40% contained spam content. As a result, bots from Campaign 4 had an average lifetime of 120 days, and started following 460 profiles each. Among these, 87 users on average followed the bots back. Campaign 7 was the most effective among Twitter campaigns, both considering the number of victims and the average bot lifetime. To achieve this, this campaign combined a low rate of messages per day with a stealth way of operating. The bots in this campaign have an average lifetime of 198 days and 1,787 victims, of which, on average, 112 acknowledged the friend request.

From the observations of the various campaigns, we developed a metric that allows us to predict the success of a campaign. We consider a campaign successful if the bots belonging to it have a long lifetime. For this metric, we introduce the parameter G_c, defined as follows:

$$G_c = \frac{M_d^{-1} \cdot S_d}{((\sqrt{M_d^{-1}} \cdot S_d) + 1)^2}, 0 \leq G_c \leq 1.$$

In the above formula, M_d is the average number of messages per day sent and S_d is the ratio of actual spam messages ($0 \leq S_d \leq 1$). Empirically, we see that campaigns with a value of G_c close to 1 have a long lifetime (for example, Campaign 7 has $G_c = 0.88$, while Campaign 2 has $G_c = 0.60$), while for campaigns with a lower value of this parameter, the average lifetime decreases significantly (Campaign 1 has $G_c = 0.28$ and

Campaign 5 has $G_c = 0.16$). Thus, we can infer that a value of 0.5 or higher for G_c indicates that a campaign has a good chance to be successful. Of course, if a campaign is active for some time, a social network might develop other means to detect spam bots belonging to it (e.g., a blacklist of the URLs included in the messages).

Activity of bots from different campaigns is shown in Figure 5.5. Each row represents a campaign. For each day in which we observed some activity from that campaign, a circle is drawn. The size of circles varies according to the number of messages observed that day. As can be seen, some campaigns have been active over the entire period of the study, while some have not been so successful.

We then tried to understand how bots choose their victims. The behavior seems not to be uniform for the various campaigns. For example, we noticed that many victims of Campaign 2 shared the same hashtag (i.e., "#iloveitwhen") in their tweets. Bots might have been crawling for people sending messages with such tag, and started following them. On the other hand, we noticed that Campaigns 4 and 5 targeted an anomalous number of private profiles. Looking at their victims, 12% of them had a private profile, while for a random picked set of 1,000 users from the public timeline, this ratio was 4%. This suggests that bots from these campaigns did not crawl any timeline, since tweets from users with a private profile do not appear on them.

5.5 Conclusions

In this chapter we showed that spam sent by fake accounts on social networks is a problem. For our study, we created a population of 900 honey-profiles on three major social networks and observed the traffic they received. We then developed SPAMDETECTOR, a system able to identify fake accounts used to spread spam, as well as large-scale campaigns. We also showed that our techniques help to detect spam profiles even when they do not contact a honey-profile. We believe that these techniques can help social networks to improve their security and detect malicious users. In fact, thousands of spamming accounts were shut down by Twitter after we reported them to their anti-spam team.

Chapter 6

Detecting Compromised Online Social Network Accounts

6.1 Introduction

In the previous chapter I introduced SPAMDETECTOR, a system to automatically detect fake accounts created to spread malicious content on online social networks. Although SPAMDETECTOR can detect fake accounts with very high accuracy, it is helpless in detecting legitimate accounts that have been compromised. A compromised account is an existing, legitimate account that has been taken over by an attacker. Accounts can be compromised in a number of ways, for example, by exploiting a cross-site scripting vulnerability or by using a phishing scam to steal the user's login credentials. Also, bots have been increasingly used to harvest login information for social networking sites on

infected hosts [123]. These credentials can then be used by miscreants to log into these accounts.

While dedicated, malicious accounts are easier to create, compromised accounts are more valuable to cyber-criminals. The reason is that these accounts allow attackers to leverage the associated account history and network of trust to spread malicious content more effectively [35]. As a result, attackers increasingly abuse legitimate accounts to distribute their malicious messages [61, 63]. To identify campaigns that involve both compromised and fake accounts, the focus of the analysis has shifted to the messages themselves. In particular, researchers have proposed techniques that search a social network for the presence of similar messages [60, 61]. The intuition is that attackers send many similar messages as part of campaigns. Similarity is typically defined in terms of overlap in message content or shared URLs.

Of course, it is not sufficient to simply group similar messages to detect malicious campaigns. The reason is that many such clusters (groups) will contain benign messages, which range from simple "happy birthday" wishes to template-based notifications sent by popular applications such as Foursquare [5]. To distinguish benign from malicious clusters, some systems utilize features that are solely based on the URLs in messages [61, 88, 137]. Clearly, these techniques suffer from limited scope, because they cannot find malicious messages that do not contain URLs (we found instances of such messages during our experimental evaluation). Other systems [60] consider additional features such as the size of clusters or the average number of connections for each user. While this broadens their coverage to include campaigns that do not rely on URLs, the reported accuracy of less than 80% is rather sobering. Moreover, and

equally important, previous systems can only determine whether a cluster of messages is malicious. That is, they *cannot* distinguish between messages sent by compromised accounts and those sent by fake accounts. This information is crucial for social network operators to initiate appropriate mitigation procedures. Specifically, fake accounts can be safely deleted without affecting legitimate users. To address compromised accounts, however, the social network provider has to notify the victims, reset their passwords, and engage the users in a password recovery process.

In this chapter I present a novel approach to detect compromised accounts on social networks. Our approach offers a combination of three salient features. First, it does not depend on the presence of URLs in messages. As a result, we can detect a broad range of malicious messages, including scam messages that contain telephone numbers and instant messaging contacts. Second, our system is accurate and detects compromised accounts with very low false positives. Third, we focus on finding compromised accounts and leave the detection of fake accounts to specialized systems such as the one we presented in Chapter 5. By identifying compromised accounts, social network providers can focus their mitigation efforts on real users.

The core idea underlying our approach is that it is possible to model the regular activities of individual users. If, at any point, a user's account gets compromised, it is likely that there will be a noticeable change in the account's behavior (and our experiments confirm this assumption). To capture the past behavior of a user, we introduce a collection of statistical models, which we call a *behavioral profile*. Each of our models corresponds to a characteristic feature of a message (e.g., the time of the day when it was sent or the language in which it was written in). These models capture generic user

activity on social networks and are not tied to a particular platform (as our experiments on Twitter and Facebook demonstrate). Behavioral profiles make it possible to assess future messages. A message that appears to be very different from a user's typical behavior might indicate a compromise.

Of course, a single message that violates the profile of a user does not necessarily indicate that this account is compromised. The message might be an outlier or merely reflect a normal change in behavior. For this reason, like in previous work, our approach looks for other, similar messages that have recently been posted on the social network and that also violate the behavioral profiles of their respective users. This means that we cannot detect cases in which an attacker posts just one malicious message through a single compromised account. We feel that this is reasonable as attackers typically aim to distribute their malicious messages to a broader victim population. Moreover, our experiments demonstrate that we can detect compromised accounts even in case of small campaigns (in our experiments on Twitter, for example, we require as little as ten similar messages per hour), and that in the very particular case in which accounts typically show consistent behaviors, such as social network profiles belonging to news agencies, our approach can detect single anomalous messages.

In a nutshell, our approach (i) checks for a set of similar messages, and (ii) requires that a significant subset of these messages violate the behavioral profiles of their senders. These two steps can be executed in any order: We can check for messages that violate their respective behavioral profiles first and then group those messages into clusters of similar ones. This would allow us to implement similarity metrics that are more sophisticated than those presented in previous work (i.e., simple checks for similar

content or URLs). Alternatively, we can first group similar messages and then check whether a substantial fraction of these messages violates the corresponding behavioral profiles. Using this order is more efficient as the system has to inspect a smaller number of messages.

We implemented our approach in a system called COMPA. Our system can be used by social network operators to identify compromised accounts and take appropriate countermeasures, such as deleting the offending messages or resetting the passwords of the victims' accounts. Since COMPA relies on behavioral patterns of users and not, like related work, on suspicious message content (URLs [137] or typical features of Sybil profiles [35]) it is able to detect types of malicious messages that are missed by recently-proposed techniques. In particular, our approach identified scam campaigns that lure their victims into calling a phone number, and hence, the corresponding messages do not contain links (URLs).

We applied COMPA to two large-scale datasets from Twitter and Facebook. The Twitter dataset consists of messages we collected from May 13, 2011 to August 12, 2011, while the Facebook dataset is the same that we used in Chapter 5. Our results show that COMPA is effective in detecting compromised accounts with very few false positives. In particular, we detected 383,613 compromised accounts on Twitter, by analyzing three months of data consisting of over 1.4 billion tweets. Furthermore, COMPA detected 11,087 compromised accounts on Facebook, by analyzing 106 million messages posted by users in several large geographic networks.

In this chapter we make the following contributions:

- We are the first to introduce an approach that focuses on detecting compromised accounts on social networks. This provides crucial input to social network providers to initiate proper mitigation efforts.

- We propose a novel set of features to characterize regular user activity based on the stream of messages that each user posts. We use these features to create models that identify messages that appear anomalous with respect to a user's account (message) history.

- We demonstrate that our approach is able to effectively detect compromised accounts with very low false positives. To this end, we applied our approach to two large-scale datasets obtained from two large social networking sites (Twitter and Facebook).

6.2 Behavioral Profiles

A behavioral profile leverages historical information about the activities of a social network user to capture this user's normal (expected) behavior. To build behavioral profiles, our system focuses on the stream of messages that a user has posted on the social network. Of course, other features such as profile pictures or friend activity could be useful as well. Unfortunately, social networks typically do not offer a way to retrieve historical data about changes in these features, and therefore, we were unable to use them.

A behavioral profile for a user U is built in the following way: Initially, our system ob-

tains the stream of messages of U from the social networking site. The message stream is a list of all messages that the user has posted on the social network, in chronological order. For different social networks, the message streams are collected in slightly different ways. For example, on Twitter, the message stream corresponds to a user's public timeline. For Facebook, the message stream contains the posts a user wrote on her own wall, but it also includes the messages that this user has posted on her friends' walls.

To be able to build a comprehensive profile, the stream needs to contain a minimum amount of messages. Intuitively, a good behavioral profile has to capture the breadth and variety of ways in which a person uses her social network account (e.g., different client applications or languages). Otherwise, an incomplete profile might incorrectly classify legitimate user activity as anomalous. Therefore, we do not create behavioral profiles for accounts whose stream consists of less than a minimum number *S* of messages. In our experiments, we empirically determined that a stream consisting of less than $S = 10$ messages does usually not contain enough variety to build a representative behavioral profile for the corresponding account. Furthermore, profiles that contain less then *S* messages pose a limited threat to the social network or its users. This is because such accounts are either new or very inactive and thus, their contribution to large scale campaigns is limited. A detailed discussion of this threshold is provided in Section 7.5.

Once our system has obtained the message stream for a user, we use this information to build the corresponding behavioral profile. More precisely, the system extracts a set of feature values from each message, and then, for each feature, trains a statistical model. Each of these models captures a characteristic feature of a message, such as the time

the message was sent, or the application that was used to generate it. The features used by these models, as well as the models themselves, are described later in this section.

Given the behavioral profile for a user, we can assess to what extent a new message corresponds to the expected behavior. To this end, we compute the anomaly score for a message with regard to the user's established profile. The anomaly score is computed by extracting the feature values for the new message, and then comparing these feature values to the corresponding feature models. Each model produces a score (real value) in the interval $[0, 1]$, where 0 denotes perfectly normal (for the feature under consideration) and 1 indicates that the feature is highly anomalous. The anomaly score for a message is then calculated by composing the results for all individual models.

6.2.1 Modeling Message Characteristics

Our approach models the following seven features when building a behavioral profile.

Time (hour of day). This model captures the hour(s) of the day during which an account is typically active. Many users have certain periods during the course of a day where they are more likely to post (e.g., lunch breaks) and others that are typically quiet (e.g., regular sleeping hours). If a user's stream indicates regularities in social network usage, messages that appear during hours that are associated with quiet periods are considered anomalous.

Message Source. The source of a message is the name of the application that was used to submit it. Most social networking sites offer traditional web and mobile web access

to their users, along with applications for mobile platforms such as iOS and Android. Many social network ecosystems provide access to a multitude of applications created by independent, third-party developers.

Of course, by default, a third-party application cannot post messages to a user's account. However, if a user chooses to, she can grant this privilege to an application. The state-of-the-art method of governing the access of third-party applications is OAUTH [10]. OAUTH is implemented by Facebook and Twitter, as well as numerous other, high-profile web sites, and enables a user to grant access to her profile without revealing her credentials.

By requiring all third-party applications to implement OAUTH, the social network operators can easily shut down individual applications, should that become necessary. In fact, our evaluation shows that third-party applications are frequently used to send malicious messages.

This model determines whether a user has previously posted with a particular application or whether this is the first time. Whenever a user posts a message from a new application, this is a change that could indicate that an attacker has succeeded to lure a victim into granting access to a malicious application.

Message Text (Language). A user is free to author her messages in any language. However, we would expect that each user only writes messages in a few languages (typically, one or two). Thus, especially for profiles where this feature is relatively stable, a change in the language is an indication of a suspicious change in user activity.

To determine the language that a message was written in, we leverage the `libtextcat`

library. This library performs n-gram-based text categorization, as proposed by Cavnar and Trenkle [42]. Of course, for very short messages, it is often difficult to determine the language. This is particularly problematic for Twitter messages, which are limited to at most 140 characters and frequently contain abbreviated words or uncommon spelling.

Message Topic. Users post many messages that contain chatter or mundane information. But we would also expect that many users have a set of topics that they frequently talk about, such as favorite sports teams, music bands, or TV shows. When users typically focus on a few topics in their messages and then suddenly post about some different and unrelated subject, this new message should be rated as anomalous.

In general, inferring message topics from short snippets of text without context is difficult. However, some social networking platforms allow users to label messages to explicitly specify the topics their messages are about. When such labels or tags are available, they provide a valuable source of information. A well-known example of a message-tagging mechanism are Twitter's *hashtags*. By prefixing the topic keyword with a hash character a user would use #Olympics to associate her tweet with the Olympic Games.

More sophisticated (natural language processing) techniques to extract message topics are possible. However, such techniques are outside the scope of this project.

Links in Messages. Often, messages posted on social networking sites contain links to additional resources, such as blogs, pictures, videos, or news articles. Links in messages of social networks are so common that some previous work has strongly focused

on the analysis of URLs, often as the sole factor, to determine whether a message is malicious or not. We also make use of links as part of the behavioral profile of a user, but only as a single feature. Moreover, recall that our features are primarily concerned with capturing the normal activity of users. That is, we do not attempt to detect whether a URL is malicious in itself but rather whether a link is different than what we would expect for a certain user.

To model the use of links in messages, we only make use of the domain name in the URL of links. The reason is that a user might regularly refer to content on the same domain. For example, many users tend to read specific news sites and blogs, and frequently link to interesting articles there. Similarly, users have preferences for a certain URL shortening service. Of course, the full link differs among these messages (as the URL path and URL parameters address different, individual pages). The domain part, however, remains constant. Malicious links, on the other hand, point to sites that have no legitimate use. Thus, messages that link to domains that have not been observed in the past indicate a change. The model also considers the general frequency of messages with links, and the consistency with which a user links to particular sites.

Direct User Interaction. Social networks offer mechanisms to directly interact with an individual user. The most common way of doing this is by sending a direct message that is addressed to the recipient. Different social networks have different mechanisms for doing that. For example, on Facebook, one posts on another user's wall; on Twitter, it is possible to directly "mention" other users by putting the @ character before the recipient's user name. Over time, a user builds a personal interaction history with other users on the social network. This feature aims to capture the interaction history for

a user. In fact, it keeps track of the users an account ever interacted with. Direct messages are sent to catch the attention of their recipients, and thus are frequently used by spammers.

Proximity. In many cases, social network users befriend other users that are close to them. For example, a typical Facebook user will have many friends that live in the same city, went to the same school, or work for the same company. If this user suddenly started interacting with people who live on another continent, this could be suspicious. Some social networking sites (such as Facebook) express this proximity notion by grouping their users into networks. The proximity model looks at the messages sent by a user. If a user sends a message to somebody in the same network, this message is considered as local. Otherwise, it is considered as not local. This feature captures the fraction of local vs. non-local messages.

If COMPA is implemented directly by a social network provider, the geo-locations of the users' IP addresses can be used to significantly improve the proximity feature. Unfortunately, this information is not available to us.

6.3 Detecting Anomalous Messages

In this section we first describe how we train our behavioral models. We then analyze the robustness of the models generated by COMPA, and evaluate the novelty of our features compared to previous work.

6.3.1 Training and Evaluation of the Models

In this section, we first discuss how we train models for each of the previously-introduced features. We then describe how we apply a model to a new message to compute an anomaly score. Finally, we discuss how the scores of the different models are combined to reach a final anomaly score that reflects how the new message is different from the historic messages used when building the model.

Training. The input for the training step of a model is the series of messages (the message stream) that were extracted from a user account. For each message, we extract the relevant features such as the source application and the domains of all links.

Each feature model is represented as a set **M**. Each element of **M** is a tuple $< fv, c >$. fv is the value of a feature (e.g., `English` for the language model, or `example.com` for the link model). c denotes the number of messages in which the specific feature value fv was present. In addition, each model stores the total number N of messages that were used for training.

Our models fall into two categories:

- *Mandatory* models are those where there is one feature value for each message, and this feature value is always present. Mandatory models are *time of the day*, *source*, *proximity*, and *language*.
- *Optional* models are those for which not every message has to have a value. Also, unlike for mandatory models, it is possible that there are multiple feature values for a single message. Optional models are *links*, *direct interaction*, and *topic*.

For example, it is possible that a message contains zero, one, or multiple links. For each optional model, we reserve a specific element with fv = `null`, and associate with this feature value the number of messages for which no feature value is present (e.g., the number of messages that contain no links).

The training phase for the *time of the day* model works slightly differently. Based on the previous description, our system would first extract the hour of the day for each message. Then, it would store, for each hour fv, the number of messages that were posted during this hour. This approach has the problem that hour slots, unlike the progression of time, are discrete. Therefore, messages that are sent close to a user's "normal" hours could be incorrectly considered as anomalous.

To avoid this problem, we perform an adjustment step after the *time of the day* model was trained (as described above). In particular, for each hour i, we consider the values for the two adjacent hours as well. That is, for each element $< i, c_i >$ of **M**, a new count $c\prime_i$ is calculated as the average between the number of messages observed during the i^{th} hour (c_i), the number of messages sent during the previous hour (c_{i-1}), and the ones observed during the following hour (c_{i+1}). After we computed all $c\prime_i$, we replace the corresponding, original values in **M**.

As we mentioned previously, we cannot reliably build a behavioral profile if the message stream of a user is too short. Therefore, the training phase is aborted for streams shorter than $S = 10$, and any message sent by those users is not evaluated.

Evaluating a new message. When calculating the anomaly score for a new message, we want to evaluate whether this message violates the behavioral profile of a user for

a given model. In general, a message is considered more anomalous if the value for a particular feature did not appear at all in the stream of a user, or it appeared only a small number of times. For *mandatory features*, the anomaly score of a message is calculated as follows:

1. The feature fv for the analyzed model is first extracted from the message. If **M** contains a tuple with fv as a first element, then the tuple $< fv, c >$ is extracted from **M**. If there is no tuple in **M** with fv as a first value, the message is considered anomalous. The procedure terminates here and an anomaly score of 1 is returned.

2. As a second step, the approach checks if fv is anomalous at all for the behavioral profile being analyzed. c is compared to $\bar{M}$, which is defined as $\bar{M} = \frac{\sum_{i=1}^{\|\mathbf{M}\|} c_i}{N}$, where c_i is, for each tuple in **M**, the second element of the tuple. If c is greater or equal than $\bar{M}$, the message is considered to comply with the learned behavioral profile for that model, and an anomaly score of 0 is returned. The rationale behind this is that, in the past, the user has shown a significant number of messages with that particular fv.

3. If c is less than $\bar{M}$, the message is considered somewhat anomalous with respect to that model. Our approach calculates the relative frequency f of fv as $f = \frac{c_{fv}}{N}$. The system returns an anomaly score of 1 - f.

The anomaly score for *optional features* is calculated as:

1. The feature fv for the analyzed model is first extracted from the message. If **M**

contains a tuple with fv as a first element, the message is considered to match the behavioral profile, and an anomaly score of 0 is returned.

2. If there is no tuple in **M** with fv as a first element, the message is considered anomalous. The anomaly score in this case is defined as the probability p for the account to have a `null` value for this model. Intuitively, if a user rarely uses a feature on a social network, a message containing an fv that has never been seen before for this feature is highly anomalous. The probability p is calculated as $p = \frac{c_{null}}{N}$. If **M** does not have a tuple with `null` as a first element, c_{null} is considered to be 0. p is then returned as the anomaly score.

As an example, consider the following check against the *language* model: The stream of a particular user is composed of 21 messages. Twelve of them are in English, while nine are in German. The **M** of the user for that particular model looks like this:

(<English,12>,<German,9>).

The next message sent by that user will match one of three cases:

- The new message is in English. Our approach extracts the tuple <English,12> from **M**, and compares $c = 12$ to $\bar{M} = 10.5$. Since c is greater than $\bar{M}$, the message is considered normal, and an anomaly score of 0 is returned.

- The new message is in Russian. Since the user never sent a message in that language before, the message is considered very suspicious, and an anomaly score of 1 is returned.

- The new message is in German. Our approach extracts the tuple <German, 9> from **M**, and compares $c = 9$ to $\bar{M} = 10.5$. Since $c < \bar{M}$, the message is considered slightly suspicious. The relative frequency of German tweets for the user is $f = \frac{c}{N} = 0.42$. Thus, an anomaly score of $1 - f = 0.58$ is returned. This means that the message shows a slight anomaly in the user average behavior. However, as explained in Section 7.5.2, on its own this score will not be enough to flag the message as malicious.

Computing the final anomaly score. Once our system has evaluated a message against each individual model, we need to combine the results into an overall anomaly score for this message. This anomaly score is a weighted sum of the values for all models. We use Sequential Minimal Optimization [105] to learn the optimal weights for each model, based on a training set of instances (messages and corresponding user histories) that are labeled as malicious and benign. Of course, different social networks will require different weights for the various features. A message is said to violate an account's behavioral profile if its overall anomaly score exceeds a threshold. In Section 7.5.2, we present a more detailed discussion on how the threshold values are determined. Moreover, we discuss the weights (and importance) of the features for the different social networks that we analyzed (i.e., Twitter and Facebook).

6.3.2 Robustness of the Models

Malicious campaigns that are executed through compromised accounts will, in general, fail to match the expected behavior of a vast majority of their victims. One reason is

that it is very difficult for an attacker to make certain features look normal, even if the attacker has detailed knowledge about the history of a victim account. In particular, this is true for the *application source* and the *links* features. Consider a user who always posts using her favorite Twitter client (e.g., from her iPhone). Since the attacker does not control this third-party application, and the social network (Twitter) automatically appends the source information to the message, a malicious message will not match the victim's history. Furthermore, to send messages from an iPhone application, the attacker would have to instrument a physical iPhone device to log into the victims' accounts and post the malicious messages. Clearly, such an attack model does not scale. To satisfy the link model, an attacker would need to host his malicious page on a legitimate, third-party domain (one of the domains that the user has linked to in the past). It is very unlikely that an attacker can compromise arbitrary third-party sites that the different victims have referenced in the past.

Other feature models can be matched more easily, assuming that the attacker has full knowledge about the history of a victim account. In particular, it is possible to post at an expected time, use a language that the victim has used in the past, and craft the message so that both the topic and direct user interactions match the observed history. However, crafting customized messages is very resource-intensive. The reason is that this would require the attacker to gather the message history for all victim users. Since social network sites typically rate-limit the access to user profiles, gathering data for many victims is a non-trivial endeavor (we initially faced similar limitations when performing our experiments; and we had to explicitly asked the social networks to white-list our IP addresses).

The need to customize messages makes it also more difficult to coordinate large-scale attacks. First, it requires delaying messages for certain victims until an appropriate time slot is reached. This could provide more time for the social network to react and take appropriate countermeasures. Second, when messages have different topics, attackers cannot easily perform search engine optimizations or push popular terms, simply because victim users might not have used these terms in the past. Also, the proximity feature can help limiting the spread of a campaign. If a user always messages users that are close to her, the number of possible victims is reduced. Of course, the burden for the attacker to blend his messages into the stream of his victims decreases with the number of victims. That is, targeted attacks against individuals or small groups are more challenging to detect. However, the precision of the behavioral profiles that COMPA generates makes us confident that similar mechanisms can contribute to the problem of identifying such small-scale targeted attacks.

Overall, given the challenges to make certain features appear normal and the practical difficulties to craft customized messages to satisfy the remaining models, our feature set is robust with regard to large-scale attacks that leverage compromised accounts. Our experiments show that COMPA is successful in identifying campaigns that use compromised accounts to distribute malicious messages.

6.3.3 Novelty of the modelled features

We also compared our features with respect to existing work. However, the purpose of our system is very different from the goals of the ones proposed in previous work. These

systems generally aim at detecting accounts that have been specifically created to spread spam or malicious content [32, 87, 130, 150]. Since these accounts are controlled in an automated fashion, previous systems detect accounts that always act in a similar way. Instead, we look for sudden changes in behavior of legitimate but compromised social network accounts. Table 6.1 lists in detail the features previous systems used to achieve their goals, and compares them to the features used by our system. In particular, we studied the works from Benevenuto et al. [32], Gao et al. [60], Grier et al. [63], Lee et al. [87], Stringhini et al. (SPAMDETECTOR from Chapter 5 [130]), Yang et al. [150], Cai et al. [40], and Song et al. [120].

As it can be seen, our system does not use any of the *Network Features* or any of the *Friends Features*. Such features aim to detect whether a certain account has been created automatically, therefore, they are not useful for our purpose. The reason is that, since the profiles we want to detect are legitimate ones that got compromised, these features would look normal for such profiles. Also, we do not use any of the *Single Message Features*. These features aim to detect a malicious message when it contains words that are usually associated with malicious content (e.g., *cheap drugs*), or when the URL is listed in a blacklist such as SURBL [14]. Since we did not want to limit our approach to flagging messages that contain known malicious sites or well-known words, we did not include such features in our models. In the future, we could use these features to improve our system.

In COMPA, we focus on *Stream Features*. These features capture the characteristics of a user's message stream, such as the ratio of messages that include links, or the similarity among the messages. Looking at Table 6.1, it seems that five of our features

(except the *Language* and *Proximity* features) have been previously used by at least one other system. However, the way these systems use such features is the opposite of what we do: Previous work wants to detect *similarity*, while we are interested in *anomalies*. For example, the *message timing feature* has been used by Grier et al. [63], by Gao et al. [60], and by COMPA for building the *time of the day* model. However, what previous work is looking for are profiles that show a high grade of automation (by looking for profiles that send messages at the same minute every hour), or for short-lived, bursty spam campaigns. Instead, we want to find profiles that start posting at unusual times.

Only the *user interactions* feature has been used in a similar fashion by another system. Gao et al. [61] use it as indication of possibly compromised accounts. Similarly to our system, they flag any account that ever had a legitimate interaction with another user, and started sending malicious content at a later time. However, they identify "malicious content" based only on URL blacklists and suspicious words in the messages. Thus, they are much more limited in their detection capabilities, and their approach mislabels fake profiles that try to be stealthy by sending legitimate-looking messages.

6.4 Grouping of Similar Messages

A single message that violates the behavioral profile of a user does not necessarily indicate that this user is compromised and the message is malicious. The message might merely reflect a normal change of behavior. For example, a user might be experimenting with new client software or expanding her topics of interest. Therefore, before we flag

	[32]	[61]	[63]	[87]	[130]	[150]	[40]	[120]	**COMPA**
Network Features									
Avg # conn. of neighbors						✔			
Avg messages of neighbors						✔			
Friends to Followers (F2F)	✔	✔			✔				
F2F of neighbors						✔			
Mutual links						✔	✔	✔	
User distance								✔	
Single Message Features									
Suspicious content	✔								
URL blacklist			✔						
Friends features									
Friend name entropy					✔				
Number of friends	✔				✔				
Profile age	✔								
Stream Features									
Activity per day	✔								
Applications used						✔			✔
Following Rate						✔			
Language									✔
Message length	✔								
Messages sent					✔				
Message similarity		✔	✔	✔	✔	✔			
Message timing		✔	✔						✔
Proximity									✔
Retweet ratio	✔								
Topics	✔								✔
URL entropy			✔						
URL ratio	✔	✔		✔	✔	✔			
URL repetition				✔					✔
User interaction	✔	✔		✔					✔

Table 6.1: Comparison of the features used by previous work.

an account as compromised, we require that we can find a number of similar messages (within a specific time interval) that also violate the accounts of their respective senders.

This means that we cannot detect cases in which an attacker posts a single, malicious message through one compromised account. While it is very possible that our models would correctly identify that message as suspicious, alerting on all behavioral profile

violations results in too many false positives. Hence, we use message similarity as a second component to distinguish malicious messages from spurious profile violations. This is based on the assumption that attackers aim to spread their malicious messages to a larger victim population. However, it is important to note that this does not limit COMPA to the detection of large-scale campaigns. In our experiments on the Twitter platform, for example, we only require ten similar messages per hour before reporting accounts as compromised.

As mentioned previously, we can either first group similar messages and then check all clustered messages for behavioral profile violations, or we can first analyze all messages on the social network for profile violations and then cluster only those that have resulted in violations. The latter approach offers more flexibility for grouping messages, since we only need to examine the small(er) set of messages that were found to violate their user profiles. This would allow us to check if a group of suspicious messages was sent by users that are all directly connected in the social graph, or whether these messages were sent by people of a certain demographics. Unfortunately, this approach requires to check *all* messages for profile violations. While this is certainly feasible for the social networking provider, our access to these sites is rate-limited in practice. Hence, we need to follow the first approach: More precisely, we first group similar messages. Then, we analyze the messages in clusters for profile violations. To group messages, we use the two simple similarity measures, discussed in the following paragraphs.

Content similarity. Messages that contain similar text can be considered related and grouped together. To this end, our first similarity measure uses n-gram analysis of a message's text to cluster messages with similar contents. We use entire words as the

basis for the n-gram analysis. Based on initial tests to evaluate the necessary computational resources and the quality of the results, we decided to use four-grams. That is, two messages are considered similar if they share at least one four-gram of words (i.e., four consecutive, identical words).

URL similarity. This similarity measure considers two messages to be similar if they both contain at least one link to a similar URL. The naïve approach for this similarity measure would be to consider two messages similar if they contain an identical URL. However, especially for spam campaigns, it is common to include identifiers into the query string of a URL (i.e., the part in a URL after the question mark). Therefore, this similarity measure discards the query string and relies on the remaining components of a URL to assess the similarity of messages. Of course, by discarding the query string, the similarity measure might be incorrectly considering messages as similar if the target site makes use of the query string to identify different content. Since `YouTube` and `Facebook` use the query string to address individual content, this similarity measure discards URLs that link to these two sites.

Many users on social networking sites use URL shortening services while adding links to their messages. In principle, different short URLs could point to the same page, therefore, it would make sense to expand such URLs, and perform the grouping based on the expanded URLs. Unfortunately, for performance reasons, we could not expand short URLs in our experiments. On Twitter, we observe several million URLs per day (most of which are shortened). This exceeds by far the limits imposed by any URL shortening service.

We do not claim that our two similarity measures represent the only ways in which

messages can be grouped. However, as the evaluation in Section 7.5 shows, the similarity measures we chose perform very well in practice. Furthermore, our system can be easily extended with additional similarity measures if necessary.

6.5 Compromised Account Detection

Our approach groups together similar messages that are generated in a certain time interval. We call this the *observation interval*. For each group, our system checks all accounts to determine whether each message violates the corresponding account's behavioral profile. Based on this analysis, our approach has to make a final decision about whether an account is compromised or not.

Suspicious groups. A group of similar messages is called a *suspicious group* if the fraction of messages that violates their respective accounts' behavioral profiles exceeds a threshold th. In our implementation, we decided to use a threshold that is dependent on the size of the group. The rationale behind this is that, for small groups, there might not be enough evidence of a campaign being carried out unless a high number of similar messages violate their underlying behavioral profiles. In other words, small groups of similar messages could appear coincidentally, which might lead to false positives if the threshold for small groups is too low. This is less of a concern for large groups that share a similar message. In fact, even the existence of large groups is already somewhat unusual. This can be taken into consideration by choosing a lower threshold value for larger groups. Accordingly, for large groups, it should be sufficient to raise an alert if a smaller percentage of messages violate their behavioral profiles. Thus, the threshold

th is a linear function of the size of the group n defined as $th(n) = \max(0.1, kn + d)$.

Based on small-scale experiments, we empirically determined that the parameters $k = -0.005$ and $d = 0.82$ work well. The max expression assures that at least ten percent of the messages in big groups violate their behavioral profiles. Our experiments show that these threshold values are robust, as small modifications do not influence the quality of the results. Whenever there are more than th messages in a group (where each message violates its profile), COMPA declares all users in the group as compromised.

Bulk applications. Certain popular applications, such as `Nike+` or `Foursquare`, use templates to send similar messages to their users. Unfortunately, this can lead to false positives. We call these applications bulk applications. To identify popular bulk applications that send very similar messages in large amounts, COMPA needs to distinguish regular client applications (which do not automatically post using templates) from bulk applications. To this end, our system analyzes a randomly selected set of S messages for each application, drawn from *all* messages sent by this application. COMPA then calculates the average pairwise Levenshtein ratios for these messages. The Levenshtein ratio is a measure of the similarity between two strings based on the edit distance. The values range between 0 for unrelated strings and 1 for identical strings. We empirically determined that the value 0.35 effectively separates client from bulk applications.

COMPA flags all suspicious groups produced by client applications as compromised. For bulk applications, a further distinction is necessary, since we only want to discard groups that are due to *popular* bulk applications. Popular bulk applications constantly recruit new users. Also, these messages are commonly synthetic, and they often violate

the behavioral profiles of new users. For existing users, on the other hand, past messages from such applications contribute to their behavioral profiles, and thus, additional messages do not indicate a change in behavior. If many users made use of the application in the past, and the messages the application sent were in line with these users' behavioral profiles, COMPA considers such an application as popular.

To assess an application's popularity, COMPA calculates the number of distinct accounts in the social network that made use of that application before it has sent the first message that violates a user's behavioral profile. This number is multiplied by an age factor (which is the number of seconds between the first message of the application as observed by COMPA and the first message that violated its user's behavioral profile). The intuition behind this heuristics is the following: An application that has been used by many users for a long time should not raise suspicion when a new user starts using it, even if it posts content that differs from this user's established behavior. Manual analysis indicated that bulk applications that are used to run spam and phishing campaigns over compromised accounts have a very low popularity score. Thus, COMPA considers a bulk application to be popular if its score is above 1 million. We assume that popular bulk applications do not pose a threat to their users. Consequently, COMPA flags a suspicious group as containing compromised accounts only if the group's predominant application is a non-popular bulk application.

6.6 Evaluation

We implemented our approach in a tool, called COMPA and evaluated it on Twitter and Facebook; we collected tweets in real time from Twitter, while we ran our Facebook experiments on a large dataset crawled in 2009.

We show that our system is capable of building meaningful behavioral profiles for individual accounts on both networks. By comparing new messages against these profiles, it is possible to detect messages that represent a (possibly malicious) change in the behavior of the account. By grouping together accounts that contain similar messages, many of which violate their corresponding accounts' behavioral profiles, COMPA is able to identify groups of compromised accounts that are used to distribute malicious messages on these social networks. We continuously ran COMPA on a stream of 10% of all public Twitter messages on a single computer (Intel Xeon X3450, 16 GB ram). The main limitation was the number of user timelines we could request from Twitter, due to the enforced rate-limits. Thus, we are confident that COMPA can be scaled up to support online social networks of the size of Twitter with moderate hardware requirements.

6.6.1 Data Collection

Twitter Dataset

We obtained elevated access to Twitter's streaming and RESTful API services. This allowed us to collect around 10% of all public tweets through the streaming API, resulting

in roughly 15 million tweets per day on average. We collected this data continuously starting May 13, 2011 until Aug 12, 2011. In total, we collected over 1.4 billion tweets from Twitter's stream. The stream contains live tweets as they are sent to Twitter. We used an observation interval of one hour. Note that since the stream contains randomly sampled messages, COMPA regenerated the behavioral profiles for all involved users every hour. This was necessary, because it was not guaranteed that we would see the same user multiple times.

To access the historical timeline data for individual accounts, we rely on the RESTful API services Twitter provides. To this end, Twitter whitelisted one of our IP addresses, which allowed us to make up to 20,000 RESTful API calls per hour. A single API call results in at most 200 tweets. Thus, to retrieve complete timelines that exceed 200 tweets, multiple API requests are needed. Furthermore, Twitter only provides access to the most recent 3,200 tweets in any user's timeline. To prevent wasting API calls on long timelines, we retrieved timeline data for either the most recent three days, or the user's 400 most recent tweets, whatever resulted in more tweets.

On average, we received tweets from more than 500,000 distinct users per hour. Unfortunately, because of the API request limit, we were not able to generate profiles for all users that we saw in the data stream. Thus, as discussed in the previous section, we first cluster messages into groups that are similar. Then, starting from the largest cluster, we start to check whether the messages violate the behavioral profiles of their senders. We do this, for increasingly smaller clusters, until our API limit is exhausted. On average, the created groups consisted of 30 messages. This process is then repeated for the next observation period.

Facebook Dataset

We tested COMPA on the same Facebook dataset that we described in Chapter 5. As we already mentioned, the dataset was crawled from geographic networks on Facebook. Geographic networks were used to group together people that lived in the same area. The default privacy policy for these networks was to allow anybody in the network to see all the posts from all other members. Therefore, it was easy, at the time, to collect millions of messages by creating a small number of profiles and join one of these geographic networks. For privacy reasons, geographic networks have been discontinued in late 2009. The dataset we used contains 106,373,952 wall posts collected from five geographic networks (i.e., London, New York, Los Angeles, Monterey Bay, and Santa Barbara). These wall posts are distributed over almost two years (Sept. 2007 - July 2009).

6.6.2 Training the Classifier

To determine the weights that we have to assign to each feature, we applied Weka's SMO [17] to a labeled training dataset for both Twitter and Facebook.

While the Facebook dataset contains the network of a user, Twitter does not provide such a convenient proximity feature. Therefore, we omitted this feature from the evaluation on Twitter. For Twitter, the weights for the features are determined from our labeled training dataset consisting of 5,236 (5142 legitimate, 94 malicious) messages with their associated feature values as follows: *Source (3.3)*, *Personal Interaction (1.4)*, *Domain (0.96)*, *Hour of Day (0.88)*, *Language (0.58)*, and *Topic (0.39)*.

To manually determine the ground truth for an account in our training set, we examined the tweets present in that account's timeline. If an account under analysis uses URLs in tweets, we follow these links and inspect the landing pages. Should we find that the URL lead to a phishing page, we classify the account as compromised. As we discuss in Section 6.6.5, phishing campaigns frequently make use of URLs to guide potential victims to phishing websites that prompt the visitor to disclose her account credentials. Another source of information we used to assess whether an account was compromised are application description pages. Each tweet sent by a third-party application contains a link to a website chosen by the developer. If such a link leads to a malicious page, we also consider the account as compromised [1]. Finally, we exploit the fact that humans can extract the topic of a message from small amounts of information. That is, we would flag an account as compromised if the topic of tweets in the timeline abruptly switches from personal status updates to tweets promoting work from home opportunities and free electronic gadgets (common scams). As we will show later in this section, a significant portion of the tweets that indicate that an account is compromised get removed. This makes it time consuming to manually identify compromised accounts on Twitter. Although the number of malicious samples in our training dataset is limited, the feature values turned out to be stable over different training set sizes.

Figure 6.1 illustrates how the weights for each feature vary with different sizes of the training set. Each set of five bars corresponds to one feature. Each bar within a set represents the observed weights for this feature (i.e., *average*, *min*, and *max*) that were produced by 25 iterations with a fixed training set size. For each iteration, the contents

[1] While Twitter has been filtering links to potentially malicious URLs in posted messages for a while, they only started filtering these application pages after *we* informed Twitter that an attacker can choose this page to be a malicious site.

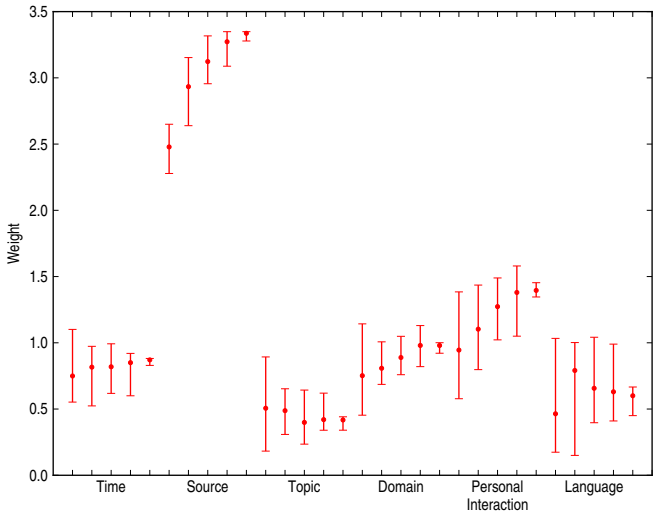


Figure 6.1: Features evolving with different sizes of training sets. Each experiment was conducted 25 times on random subsets of 25%, 50%, 70%, 90%, and 99% of the 5,236 labeled training instances. The fraction of positive to negative samples remained constant.

of the training set were randomly chosen. Overall, this experiment was repeated five times with different training set sizes. It can be seen that when smaller training sets are used, the observed weights vary heavily. This variance becomes small for lager training datasets indicating that the weights are fairly stable.

On Facebook, based on a labeled training dataset of 279 messages (181 legitimate, 122 malicious), the weights were: *Source (2.2)*, *Domain (1.1)*, *Personal Interaction (0.13)*, *Proximity (0.08)*, and *Hour of Day (0.06)*. Weka determined that the *Language* feature has no effect on the classification. Moreover, as discussed earlier, assessing the message topic of an unstructured message is a complicated natural language processing problem. Therefore, we omitted this feature from the evaluation on the Facebook dataset. Similar to analyzing Twitter messages, we also assessed changes of topic across wall posts in

the Facebook dataset to identify compromised accounts for the training data. Additionally, we inspected Facebook application pages and their comment sections where users can leave feedback. As the Facebook dataset was collected in 2009, we would also consider an account as compromised if the application that sent that post was blocked by Facebook in the meantime.

6.6.3 Detection on Twitter

The overall results for our Twitter evaluation are presented in Table 6.2. Due to space constraints, we will only discuss the details for the *text similarity measure* here. However, we found considerable overlap in many of the groups produced by both similarity measures. More precisely, for over 8,200 groups, the two similarity measures (content and URL similarity) produced overlaps of at least eight messages. COMPA found, for example, phishing campaigns that use the same URLs and the same text in their malicious messages. Therefore, both similarity measures produced overlapping groups.

The *text similarity* measure created 374,920 groups with messages of similar content. 365,558 groups were reported as legitimate, while 9,362 groups were reported as compromised. These 9,362 groups correspond to 343,229 compromised accounts. Interestingly, only 12,238 of 302,513 applications ever produced tweets that got grouped together. Furthermore, only 257 of these applications contributed to the groups that were identified as compromised.

For each group of similar messages, COMPA assessed whether the predominant application in this group was a regular client or a bulk application. Our system identi-

Network & Similarity Measure	Twitter Text		Twitter URL		Facebook Text	
	Groups	Accounts	Groups	Accounts	Groups	Accounts
Total Number	374,920		14,548		48,586	
# Compromised	9,362	343,229	1,236	54,907	671	11,499
False Positives	4% (377)	3.6% (12,382)	5.8% (72)	3.8% (2,141)	3.3% (22)	3.6% (412)
# Bulk Applications	12,347		1,569		N/A	N/A
# Compromised Bulk Applications	1,647	178,557	251	8,254	N/A	N/A
False Positives	8.9% (146)	2.7% (4,854)	14.7% (37)	13.3% (1,101)	N/A	N/A
# Client Applications	362,573		12,979		N/A	N/A
# Compromised Client Applications	7,715	164.672	985	46,653	N/A	N/A
False Positives	3.0% (231)	4.6% (7,528)	3.5% (35)	2.2% (1,040)	N/A	N/A

Table 6.2: Evaluation Results for the Text (Twitter and Facebook) and URL (Twitter) Similarity measure.

fied 12,347 groups in the bulk category, of which 1,647 were flagged as compromised. Moreover, COMPA identified a total of 362,573 groups that originated from client applications. Of these, 7,715 were flagged as compromised.

Overall, our system created a total of 7,250,228 behavioral profiles. COMPA identified 966,306 messages that violate the behavioral profiles of their corresponding accounts. Finally, 400,389 messages were deleted by the time our system tried to compare these messages to their respective behavioral profiles (i.e., within an hour).

False Positives

Using the text similarity measure, COMPA identified 343,229 compromised Twitter accounts in 9,362 clusters. To analyze the accuracy of these results, we need to answer two questions: First, do we incorrectly label non-compromised accounts as compromised (false positives)? We try to answer this question in this subsection. Second, do we miss accounts that have been compromised (false negatives)? We discuss this question in the next subsection.

False positives might arise in two cases: First, legitimate users who change their habits (e.g., a user experiments with a new Twitter client) might be flagged as compromised. Second, fake accounts, specifically created for the purpose of spreading malicious content, might trigger our detection, but these are not compromised accounts. Arguably, the second source of false positives is less problematic than the first one, since the messages that are distributed by fake accounts are likely malicious. However, since social network providers need to handle compromised accounts differently from fake accounts (which can be simply deleted), we want our system to only report compromised accounts.

To address the first reason for false positives, we analyzed the groups that our similarity measures generated. First, we aggregated similar (repeated) groups into long-lasting campaigns. Two groups belong to the same campaign if all pairwise Levenshtein ratios between ten randomly-chosen messages (five messages from each group) is at least 0.8. We could aggregate 7,899 groups into 496 campaigns. We then manually analyzed a subset of the accounts present for each campaign. Additionally, 1,463 groups did not belong to any campaign, and we assessed each of these manually. During manual analysis, we opted to err on the conservative side by counting groups whose accounts contain messages written in non-Latin based languages as false positives.

In total, 377 of the 9,362 groups (4%) that COMPA flagged as containing compromised accounts could not be verified as such, and thus, constitute false positives. Note that each group consists of multiple tweets, each from a different Twitter account. Thus, the above mentioned results are equivalent to flagging 343,229 user as compromised, where 12,382 (3.6%) are false positives.

Three months after we finished our experiments, we tried to retrieve all messages that we found were indicative of compromised accounts. Only 24% were still available. Furthermore, we would expect that the majority of messages sent by legitimate accounts are persistent over time. Thus, we also tried to retrieve a random sample of 160,000 messages contained in clusters that COMPA found to be benign. 82% of these messages were still reachable. Additionally, we also tried to access 64,368 random messages that we collected during our experiments as described in the following subsection. Of these, 84% were still accessible.

This means that for 76% of the messages that COMPA identified as being sent by a compromised account, either Twitter or the user herself removed the message. However, 96.2% of the accounts that sent these tweets were still accessible. For less than one percent (0.6%) of the accounts, Twitter states that they were suspended. The remaining 3.2% return a "Not found" error upon access. These percentages are almost perfectly in line with accounts that COMPA did not flag as compromised (95.5%, 0.5%, and 4%, respectively), and a random sample of 80,000 accounts (94%, 4%, and 2%). Twitter actively suspends spam accounts on their network. Thus, these results indicate that Twitter does not consider the accounts flagged by COMPA as fake. However, the significant amount of removed messages for such accounts leads us to believe that COMPA was successful in detecting compromised accounts.

To estimate the second source of false positives, we tested our results against SPAMDETECTOR. This allows us to detect accounts that were fake accounts as opposed to compromised. As we explained in Chapter 5, SPAMDETECTOR detects fake accounts that are likely spammers, based on features related to automatically created and managed

accounts (such as the ratio between the friend requests that are sent and the requests that are accepted, the fraction of messages with URLs, and how similar the messages are that a single user sends). This system proved to be effective in detecting accounts that have been specifically created to spread malicious content. However, the system does not usually detect compromised accounts. This is because such accounts usually have a long history of legitimate Twitter usage, and, hence, the few tweets that are sent out after the compromise do not affect the features their classifier relies on.

We used this classifier to analyze a random sample of 94,272 accounts that COMPA flagged as compromised. The idea is that any time an account is detected as a spammer, this is likely to be an account specifically created with a malicious intent, and not a legitimate, yet compromised account. Out of the analyzed accounts, only 152 (0.16%) were flagged as spammers. We then manually checked these accounts to verify if they were actually false positives of COMPA. 100 of these 152 accounts turned out to be compromised, and thus, true positives of COMPA. The reason for flagging them as spam accounts is that they have not been very active before getting compromised. Therefore, after the account was compromised, the spam messages had more influence on the features than the legitimate activity before the compromise. The remaining 52 accounts were not compromised but had been specifically created to spam. However, these 52 accounts were distributed over 34 clusters with an average cluster size of 30. Furthermore, no cluster consisted solely of false positives. The main reason why they were detected as behavior violations by COMPA is that they posted an update in an hour during which they had never been active before. This result underlines that the compromised accounts that COMPA reports are substantially different than the dedicated, fake accounts typically set up for spamming.

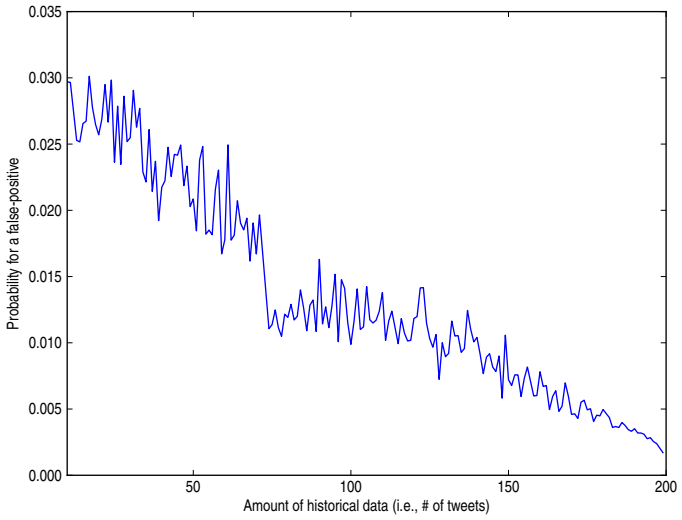


Figure 6.2: Probability of false positives depending on the amount of historical data on Twitter.

Historical information. We also investigated how the length of a user's message stream influences the quality of the behavioral profiles COMPA builds (recall that COMPA does not build a behavioral profile when a user has posted fewer than 10 messages). To this end, we calculated the probability of a false positive depending on the number of tweets that were available to calculate the behavioral profile. As Figure 6.2 illustrates, COMPA produces less false positives for accounts whose historical data is comprehensive. The reason for this is that the models become more accurate when more historical data is available.

False Negatives

To assess false negatives, we used COMPA to create 64,368 behavioral profiles for randomly selected users over a period of 44 days. To this end, every minute, COMPA

retrieved the latest tweet received from the Twitter stream and built a behavioral profile for the corresponding user. 2,606 (or 4%) of these profiles violated their account's behavioral profile. 415 of these were sent by known, popular bulk applications. We manually inspected the remaining 2,191 tweets that violated their accounts' behavioral profiles (we performed the same manual analysis that was previously used to determine the ground truth for our training set). We did not find evidence of any malicious activity that COMPA missed.

In a next step, we extracted all URLs posted on Twitter during one day, and we checked them against five popular blacklists. The idea behind this is that if a URL is known to be malicious, it is likely to be posted either by a compromised or by a fake account. We extracted 2,421,648 URLs (1,956,126 of which were unique) and checked them against the *Spamhaus Domain Blacklist* [13], *Google Safebrowsing* [6], *PhishTank* [11], *Wepawet* [18], and *Exposure* [3]. We first expanded shortened URLs before checking the landing page against the blacklists. In total, 79 tweets contained links that were present in at least one blacklist (these 79 tweets contained 46 unique URLs). We ran COMPA on each of the 79 messages to see if they were actually sent by compromised accounts.

Our system flagged 33 messages as violating their user's profile. The reason COMPA did not flag these accounts in the first place is that the clusters generated by these messages were too small to be evaluated, given the API limit we mentioned before. If we did not have such a limit, COMPA would have correctly flagged them. Seven more messages contained URLs that were similar to those in the 33 messages. Even though these compromised accounts did not violate their behavioral profiles, they would have

been detected by COMPA, because they would have been grouped together with other messages that were detected as violating their behavioral profiles.

Of the remaining 39 accounts that COMPA did not flag as compromised, 20 were detected as fake accounts by the classifier by SPAMDETECTOR, and are therefore considered as fake accounts. We manually investigated the remaining 19 results. 18 of them contained links to popular news sites and blogs, which were mainly blacklisted by Google Safebrowsing. We think users posted legitimate links to these pages, which might have become compromised at a later point in time (or are false positives in Google Safebrowsing). Thus, we do not consider accounts that linked to such pages as either compromised or fake.

The remaining message linked to a phishing page, but did not violate the profile of the account that posted it. We consider this as a message by a compromised account, and, therefore, a false negative of COMPA.

6.6.4 Detection on Facebook

As the Facebook dataset spans almost two years we increased the *observation interval* to eight hours to cover this long timespan. Furthermore, we only evaluated the Facebook dataset with the text similarity measure to group similar messages.

Our experiments indicated that a small number of popular applications resulted in a large number of false positives. Therefore, we removed the six most popular applications, including *Mafia Wars* from our dataset. Note that these six applications resulted

in groups spread over the whole dataset. Thus, we think it is appropriate for a social network administrator to white-list applications at a rate of roughly three instances per year.

In total, COMPA generated 206,876 profiles in 48,586 groups and flagged 671 groups as compromised (i.e, 11,499 compromised accounts). All flagged groups are created by bulk applications. 22 legitimate groups were incorrectly classified (i.e., 3.3% false positives) as compromised; they contained 412 (3.6%) users.

6.6.5 Case Studies

In this section, we describe some interesting findings about the compromised accounts detected by COMPA.

"Get more Followers" scams On Twitter, the majority of the accounts that COMPA flagged as compromised were part of multiple large-scale phishing scams that advertise "more Followers". These campaigns typically rely on a phishing website and a Twitter application. The phishing website promises more followers to a user. The victim can either get a small number of followers for free, or she can pay for a larger set of followers. Many users consider the number of their followers as a status symbol on the Twitter network, and the "base version" of the service is free. This combination seems to be an irresistible offer for many. The phishing sites requires the user to share their username and password with the website. Additionally, the user needs to give read and write access to the attacker's application. Once the victim entered her credentials and

authorized the application, the application immediately posts a tweet to the victim's account to advertise itself. Subsequently, the attackers make good on their promise and use their pool of existing, compromised accounts to follow the victim's account. Of course, the victim also becomes part of the pool and will start following other users herself.

COMPA identified four different web sites that were part of the same phishing campaign. [2] Although the web sites look different, they all use the same application to post tweets to their victim's accounts. This behavior became evident when we combined the results of the *text similarity* and *landing page* oracles. More precisely, the *landing page* oracle identified four distinct campaigns, whereas the *text similarity* oracle detected a single campaign.

Among the others, COMPA successfully identified four different phishing campaigns. Although these campaigns were sending messages pointing to different URLs, the structure of the messages and the application used was the same. This means that the same group of people is running these campaigns and the infrastructure behind them. Moreover, this gives an idea of how sophisticated the underground economy behind Twitter has become.

Victim analysis. We also analyzed the victims of one of the "Get more followers" scams previously described. The goal of this analysis was to understand more about the victims who fall for such scams. Over the analysis period, COMPA detected 84,650 accounts that were compromised by this campaign. All together, these accounts had

[2] `http://plusfollower.info`, `http://followback.info`, `http://hitfollow.info`, and `http://newfollow.info`

4,249,788 friends. We started to look for profiles that were followed by a large number of victims. The idea behind this is that this type of scams is likely to produce dense clusters of users connected to each other, since the advertised goal of the campaign is to give more followers to the accounts that subscribe to the service. The easiest way of doing this is to make the accounts attackers obtained credentials to follow each other. Overall, 51,584 profiles were followed by at least 500 victims of the scam. However, the majority of these accounts belonged to celebrities, which are followed by hundreds of thousands of twitter accounts. To discriminate between celebrities and other profiles, we used the *Klout* API [81]. Klout is a service that measures how influential a Twitter account is. By leveraging their API, we identified which accounts belonged to celebrities, and which to users that were not influential but still had an anomalous number of scam victims following them. In particular, we considered any profile with a Klout score above 50 to be an influential profile. We found 204 accounts had both a low Klout score and a high number of victims following them. We crawled the timelines of these accounts to assess whether they were also compromised. 22 of them sent tweets belonging to the scam, and therefore we consider them as compromised. Interestingly, 182 profiles did not show a behavior typical of compromised profiles. However, due to their low influence scores, we think it is unlikely that they got followed by such a large number of users who also happened to be compromised by the same scam. This fact suggests that not only the victims of the scam will be followed by other compromised accounts, but also that such victims will massively follow other accounts that are not popular on Twitter. This might mean that this particular scam is related to a services that offers followers in exchange for money. In fact, we observed such campaigns that would, for example advertise 3,000 followers for $65. The scammers make good

on their offer by following their customers with accounts for which they previously phished the credentials.

In exchange for the user's money the scammers follow their customers with a number of profiles for which they previously phished the credentials. A more detailed analysis of this type of scams has been presented in our follow-up papers [127, 132].

Phone numbers COMPA also detected scam campaigns that do not contain URLs. Instead, potential victims are encouraged to call a phone number. Such messages would read, for example, "Obama is giving FREE Gas Cards Worth $250! Call now-> 1 888-858-5783 (US Only)@@@." In our evaluation, 0.3% of the generated groups did not include URLs. Existing techniques, such as [137], which solely focus on URLs and their reputation, would fail to detect such campaigns.

Malicious Firefox plugin Furthermore, COMPA detected a campaign to distribute a malicious Firefox plugin under the false premise that the plugin enables the non-existing *dislike button* on Facebook. However, once installed the plugin would annoy the user with additional advertisements.

6.6.6 Detecting Worms

Twitter has been affected by multiple worm outbreaks. For example, in September 2010 a XSS worm exploited a vulnerability in the way in which Twitter parsed URLs in tweets. More specifically, if a URL contained an "@" symbol, Twitter would interpret

everything following that character as JavaScript. Therefore, a user who hovered her mouse over a tweet containing a URL similar to

```
http://x.xx/@"onmouseover="alert(1)
```

would execute the JavaScript event handler in her browser. Of course, the real worm used JavaScript that would self propagate the worm instead of the `alert` statement. Obviously, posting the tweet that contained the body of the worm happened without the user's consent, and, therefore, we have to consider such accounts as compromised. Note that the URL preceding the @ sign was irrelevant for the attack. Therefore, existing detection approaches that examine the maliciousness of URLs would fail to detect this XSS worm attack, as the attacker could chose any benign domain (e.g., `http://www.google.com`).

To evaluate whether COMPA is capable of detecting worm outbreaks, we simulated the worm outbreak on real Twitter data. That is, we chose a random message S_0 of a random user U_0 on the Twitter network. We assumed that the worm would propagate from user *A* to user *B* iff user *B* follows user *A*, and user *B* was active on Twitter within a time window *T* around the point in time when user *A* posts the offending message. Due to the lack of detailed usage information, we determine the activity of a user by observing when they tweet. Thus, a user is deemed active *T*/2 before and after she posted any status updates through the Twitter web interface. Note that this definition of activity (i.e., a user is only deemed active when she is posting) is conservative, as users are often browsing Twitter or reading other people's tweets, even if they do not post at the same time. Furthermore, the worm only propagates itself if the tweet that user *B*

sent was posted through the Twitter web site. That is, alternative clients are assumed not to contain the same vulnerability in their URL parsing routines. The XSS worm we are simulating is aggressive in that it spreads as soon as the user hovers the mouse over the tweet. We assume that if a user is active, she will hover her mouse over the tweet, and thus, get infected. For every propagation step, we record the IDs of users *A* and *B*, as well as the ID of the tweet that was used to determine that user *B* is active (i.e., the tweet user *B* sent within the time window *T*). According to [9], web users spend roughly 10 minutes per day on social networks. Thus, we assumed a value of 10 minutes for *T* in our simulation.

Subsequently, we downloaded the timelines of the users infected by the simulated worm. Then, we substituted the tweets that were responsible for the worm propagation with a copy of the XSS worm. Finally, we ran COMPA on these timelines. Although the way we simulated the worm outbreak means that the timing and source models are drawn from real information (i.e., we only substituted the text of the tweet), COMPA was able to successfully detect the outbreak and the compromised accounts after the worm spread to 2,256 accounts in 20 minutes. This means that the "worm group" contained enough tweets that violated their respective users' behavioral profiles. It turns out that our propagation strategy was chosen conservatively as news reports[3] of previous Twitter worms report of 40,000 infected accounts within 10 minutes. Thus, assuming the distribution of profile violations is similar for such aggressive worms, COMPA would detect such a large scale outbreak even faster.

[3] http://eu.techcrunch.com/2010/09/21/warning-mouseover-tweets-security-flaw-is-wreaking-havoc-on-twitter/

6.7 Detecting High-profile Compromises

As we mentioned in Section 6.4, it is not uncommon for regular Twitter users to generate a tweet that violates their behavioral profile. For this reason, COMPA groups together similar messages, and flags a group as malicious if a relevant fraction of the messages result suspicious compared to their respective behavioral profile. By doing this, we are able to reliably detect large-scale compromises that take over multiple social network accounts and use them for malicious purposes.

Although large-scale compromises account for the majority of compromised accounts on social networks, there is another class of compromises that is possibly more dangerous: high-profile accounts, belonging to celebrities, corporations, and news agencies can be hijacked by miscreants and used to spread false information. Since these high-profile accounts have a good reputation, the public is likely to believe this false information, with potentially negative effects.

In this section we investigate the ability of COMPA to detect high-profile compromises that target a single account and are composed of a single message. Our intuition is that these high-profile accounts will show a more consistent behavior than regular social network accounts, and therefore COMPA can reliably detect anomalies for them with very low false positives. To validate this intuition, we analyzed the behavior of the Twitter accounts of 20 high-profile news agencies over time. To this end, we collected the public timelines of these 20 accounts and performed the behavioral profile extraction and evaluation presented earlier in this chapter for the most recent 500 tweets in these timelines. Since we assume that the intention of media accounts is to dissemi-

Twitter Account	#Violations (%)	Twitter Account	#Violations (%)
FoxNews	0 (0%)	washingtonpost	1 (0%)
CNN	0 (0%)	lemondefr	2 (0%)
foxnewspolitics	0 (0%)	Reuters	5 (1%)
AP	0 (0%)	BostonGlobe	11 (2%)
latimes	0 (0%)	BBCNews	16 (3%)
BloombergNews	0 (0%)	msnbc	17 (3%)
HuffingtonPost	0 (0%)	NBCNews	19 (4%)
BW	0 (0%)	el_pais	22 (4%)
abcnews	1 (0%)	DerSPIEGEL	63 (13%)
nytimes	1 (0%)	guardian	100 (20%)

Table 6.3: Behavioral profile violations of news agency Twitter accounts within most recent 500 tweets.

nate original content, our analysis discards all retweets and conversations (i.e., replies) contained in a timeline. Our analysis has shown that such interaction frequently diverts from observed regular behavior and is only used in exceptional cases by official news agency Twitter accounts.

We consider an account to show consistent behavior if the probability of any of its messages violating the accounts behavioral profile is significantly less than 4%. As we mentioned previously, this is the rate of messages that randomly violates the behavioral profile of an average Twitter account. As Table 6.3 illustrates, the rate of messages violating the behavioral profile for the majority of the media accounts that we analyzed is very low. This is an indicator that COMPA could be used to protect high-profile Twitter account against compromises. In the following, we discuss these results more in detail, and provide two examples of high-profile compromises that COMPA would have been able to detect.

Analysis The only two media accounts whose behavior was not vastly consistent are the accounts of the German news outlet *Der Spiegel* and the British *Guardian*. Most profile violations in the @*derspiegel* account arose because this account is not only used to disseminate news but also to interact with other reporters. For example, advertising and communication during a journalistic convention held at the Spiegel headquarters are also included in that timeline. We suspect that the Guardian Twitter feed is managed by a set of different actors who have different preferences in terms of Twitter clients and slightly different editing styles. Our system is currently not able to characterize accounts with such multi-variant behavior patterns.

In the following we analyze two attacks on high-profile media accounts in more detail. Our previous research found that accounts are frequently compromised by tricking the user into authorizing malicious Twitter applications. In contrast, the two attacks that we detail here share the characteristic that the attackers used the Twitter web interface to distribute fake news. One can only log into a Twitter account if the account name and password are known. Thus, we can conclude that the attackers in these cases had full access to the compromised accounts allowing them to authorize additional applications or change the accounts password. These actions require full access to the account and cannot be performed if the attacker only tricked the user to authorize a malicious application. A detailed report [23] from another victim of a similar attack supports this assumption by detailing the phishing emails used.

Associated Press On April 23^{rd} 2013, the Twitter account of the Associated Press (@AP) was compromised [16]. The account was misused to distribute false information

about president Obama being hurt by an explosion in the White House. Comparing the behavioral profile of the @AP account against this message resulted in significant differences among many features that our system evaluates. For example, the fake news was posted via the Twitter website, whereas the legitimate owners of the @AP account commonly use the SocialFlow application to send status updates. Furthermore, the fake tweet did not include any links to additional information, a practice that the @AP account follows very consistently.

Only two features in our behavioral model did not signify a change of behavior. The time when the tweet was sent (i.e., 10:07UTC) and the language of the tweet itself. The authors of the @AP account as well as the attackers used the English language to author their content. While the language is undoubtedly the same, a more precise language analysis could have determined an error in capitalization in the attacker's message.

FoxNews Politics On July 4^{th} 2011, the Twitter account of Fox News' politics (@foxnewspolitics) division got compromised [20]. The attackers used this opportunity to distribute the information that president Obama got assassinated. This tweet violated almost all the features used by our system. For example, the tweet was sent in the middle of the night (i.e., 23:24UTC), through the main Twitter web site. Furthermore, it did not include a link to the full story on the Fox News website. The tweet also made extensive use of hashtags and mentions, a practice not commonly used by the @foxnewspolitics account.

6.8 Limitations

An attacker who is aware of COMPA has several possibilities to prevent his compromised accounts from being detected by COMPA. First, the attacker can post messages that align with the behavioral profiles of the compromised accounts. As described in Section 6.3, this would require the attacker to invest significant time and computational resources to gather the necessary profile information from his victims. Furthermore, social networks have mechanisms in place that prevent automated crawling, thus slowing down such data gathering endeavors.

Second, an attacker could send messages that evade our similarity measures, and thus, although such messages might violate their compromised accounts' behavioral profiles, they would not get grouped together. To counter such evasion attempts, COMPA can be easily extended with additional and more comprehensive similarity measures. For example, it would be straight-forward to create a similarity measure that uses the landing page instead of the URLs contained in the messages to find groups of similar messages. Furthermore, more computationally expensive similarity measures, such as text shingling or edit distances for text similarity can also be implemented. Other similarity measures might leverage the way in which messages propagate along the social graph to evaluate message similarity.

6.9 Conclusions

In this chapter, we presented a novel approach to detect compromised accounts in social networks. More precisely, we developed statistical models to characterize the behavior of social network users, and we used anomaly detection techniques to identify sudden changes in their behavior. We developed COMPA, a prototype tool that implements this approach, and we applied it to a large stream of messages. The results show that our approach reliably detects compromised accounts, even though we do not have full visibility of every message exchanged on Facebook and Twitter.

Part III

Detecting The Relations Between Malicious Hosts and Online Accounts

Chapter 7

Detecting Malicious Account Communities on Online Services

7.1 Introduction

As we mentioned in Chapter 1.1, attackers need two resources to carry out malicious campaigns on online services: *online accounts* and *infected machines*. Almost all online services require users to sign up and create accounts before they can access the functionality that these services offer. Accounts allow online services to associate data with users (such as emails, posts, pictures, ...), and they also serve as a convenient way to regulate and restrict access. Infected machines (bots) are the typical mean through which attackers access online accounts. They are the devices (hosts) that run the clients that allow the miscreants to connect to online services. Infected machines work well for

cybercriminals because they serve as a convenient way for the attacker to log into the targeted service and issue the necessary commands to send spam or harvest personal information of legitimate users. However, attackers do not necessarily have to leverage infected machines to connect to online services. They could also use compromised servers, or even the cybercriminal's personal device. For this reason, to keep our analysis generic in this chapter we refer to any device used by a cybercriminal to an online account as a *connection point*.

In the previous chapters of this thesis we focused on systems that either detect infected computers or malicious accounts on online services. In this chapter instead we propose a novel detection approach based on the analysis of the interactions between attackers and an online service. More precisely, we look at the interplay between accounts, connection points, and actions. That is, we observe which account carries out what action, and which connection point is responsible for originating that action.

The basis for our detection approach is the intuition that cybercriminals use online services differently than regular users. Cybercriminals need to make money, and this requires automated operations at a large scale. Thus, when such operations are carried out, they involve many accounts, connection points, and actions. Moreover, accounts and connection points are related in interesting ways that can be leveraged for detection. A key reason for these interesting relationships is the fact that attackers use botnets (as connection points) to access the online accounts that participate in a campaign. By linking accounts and the connection points that are used to access these accounts, we see that malicious *communities* emerge, and these communities can be detected.

Our approach works by identifying communities (sets) of online accounts that are all

accessed from a number of shared connection points (we use IP addresses to identify these connection points). That is, we observe a number of IP addresses and accounts, and each account is accessed by a non-trivial portion of these IP addresses. Typically, these IP addresses correspond to bot-infected machines, and they are used to log into the accounts that are under the control of the attacker. To identify communities, we consume a log of *interaction events* that the online services record. An interaction event can be any action that a user performs in relation to an account on an online service, such as logging in, sending an email, or making a friend request. Each event also contains the account that is involved, as well as the IP address that sends the request.

In a next step, we analyze the characteristics of accounts within a community and identify typical behaviors that are indicative of malicious activity. Such characteristics include suspicious activity frequencies over time, synchronized activity of the machines using the accounts in the community and the distribution of the types of browsers used by the infected machines to connect to the online accounts. Analyzing additional characteristics allows us to identify communities of accounts that are used for legitimate purposes, and, thus, reduce false positives. Interestingly, our results show that the overwhelming majority of accounts that are part of communities are actually malicious. Hence, we do not need to leverage additional characteristics for detection. Instead, we explore them to shed light onto the operations of cybercriminals on online services.

One advantage of our approach is that it is very general. We do not leverage specific information of a particular online service, and our definition of interaction events can be very broad. In fact, we show that our approach works both for the detection of spammers on a webmail service, as well as for the identification of malicious accounts

on social networks. In the former case, interaction events correspond to the sending of emails, while in the latter case, an interaction event is recorded when a user logs into her account. This is different from previous work that looked at malicious activity on online services. For instance, our approach can be applied to different types of actions. These actions can include account generation and login operations. In these cases, it might be possible to detect malicious accounts before they distribute any malicious content, as an early warning system. Also, it can help to identify abuses where no malicious content is spread. An example of this are botnets that use online social networks as part of their command-and-control (C&C) infrastructure [123], or botnets that crawl the address books of webmail users looking for victim email addresses [1].

We implemented our approach in a system called EVILCOHORT. We evaluated EVILCOHORT on real-world data collected from five different online services, and monitored the communities of online accounts that were detected. Over a period of five months, EVILCOHORT detected more than one million online accounts as malicious.

In summary, this chapter makes the following contributions:

- We find that a significant amount of malicious activity is carried out by accounts that form communities (when looking at the connection points that access them). We also find that these accounts are harder to detect for online services and remain active longer.
- We present an approach to detect malicious communities (and hence, accounts controlled by cybercriminals). This approach works by detecting accounts that are accessed by a common, shared set of IP addresses.

- We implemented our approach in a system called EVILCOHORT. We evaluated EVILCOHORT on datasets of different types of interactions collected on five different online services. Over a period of five months, EVILCOHORT detected more than one million accounts used to perform malicious activity. We show that EVILCOHORT is effective in detecting malicious online communities regardless of the type of accounts analyzed, making it a valuable tool to protect a variety of online services from being abused.

- We analyzed the communities detected by EVILCOHORT. We show that malicious account communities have very peculiar characteristics, which could support automated detection but also prevention of false positives.

7.2 Background: Analysis of Malicious Activity on a Webmail Service

We want to understand the way in which cybercriminals abuse accounts on online services, to identify weak points that we could leverage for detection. To this end, we observed the email-sending activity on a large webmail service for the period of one day. Our dataset was composed of a random sample of 72,471,992 emails generated by 21,387,006 distinct online accounts. We call the dataset containing information about this email-sending activity **T**. For each email-sending event, the dataset **T** contains the IP address that accessed the account, the user ID of the account that sent the email, and a timestamp. In addition, each email-sending event contains information on whether the email was considered as spam by the webmail provider or not. The webmail provider

uses a variety of anti-spam techniques, from IP reputation to content analysis.

Two Types of Malicious Accounts. We analyzed the accounts that sent spam in the dataset **T**. We identify two types of malicious accounts:

1. *Accounts that are used in isolation*. In these operations, each account is accessed by a single IP address, which could be the attacker's computer or a single infected machine.

2. *Accounts that are accessed by multiple IP addresses*. In these operations, the same account is accessed by multiple infected computers (i.e., by different IP addresses).

We looked at how many malicious accounts of each type are active on the webmail service. We considered an account as malicious if the account sent at least 10 emails during the day under consideration, and 60% or more of these emails were flagged as spam by the webmail provider. We selected this threshold because we needed a set of "labeled" accounts that sent spam on the webmail provider. Picking accounts whose majority of emails was flagged as spam by the email provider gives us confidence that this dataset does not contain false positives. We call this set of labeled accounts **L**. In total, **L** is composed by 66,509 malicious accounts were accessed by a single IP address, and 103,918 malicious accounts that were accessed by two or more IP addresses.

Accounts Shared by Many IP Addresses Are More Dangerous. We then investigated the effectiveness of the two identified types of spam accounts in sending emails, and

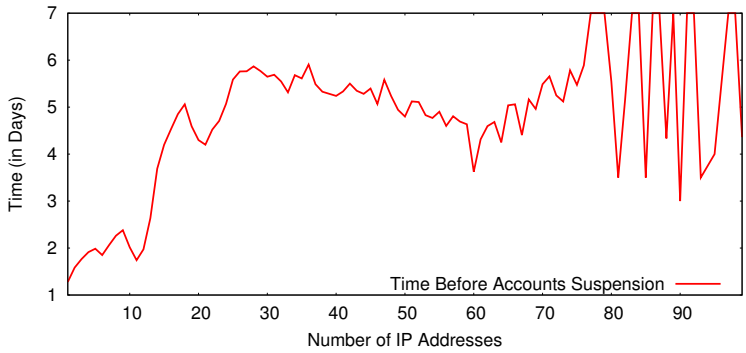


Figure 7.1: Average time (in days) before a spamming account was suspended in **L**, given the number of IP addresses accessing that account. Accounts accessed by many IP addresses can be stealthier in their operation and survive longer. The plot shows bumps when we reach accounts accessed by a very high number of accounts, because these accounts are very rare.

their ability to evade detection by the webmail provider. Figure 7.1 shows the average time (in days) that it took for a malicious account in **L** to be suspended after it sent the first spam email, given the number of IP addresses that accessed that account. As it can be seen, accounts that are used in isolation have a shorter lifespan than the ones that are used by multiple IP addresses: accounts that are only accessed by a single IP address are typically detected and suspended within a day, while ones that are accessed by many different IPs can survive for as long as a week.

We then studied the difference in the activity of the two types of accounts with regards to the number of spam emails sent, to identify techniques that might allow us to detect them more effectively. Figure 7.2 shows that accounts that are used in isolation are less effective for cybercriminals, as they send a smaller number of emails per day before being shut down. Alternatively, attackers can have each of their infected computers send a small number of emails and stay under the radar. Figure 7.3 shows that IP addresses accessing accounts used in isolation send 19 emails per day on average before

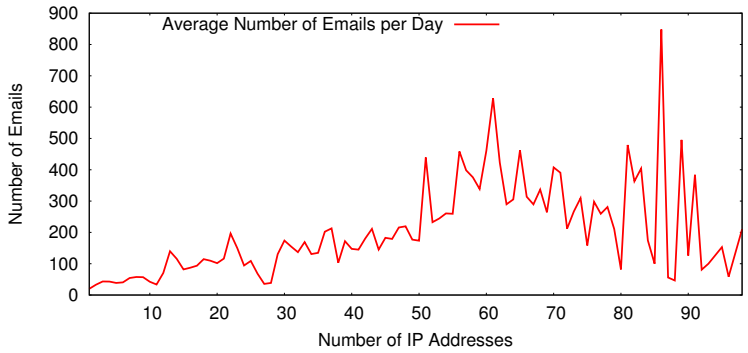


Figure 7.2: Number of spam emails sent per day on average by accounts accessed by a certain number of IP addresses. Accounts accessed by more IP addresses are generally able to send more emails from their malicious accounts.

being blocked, while having multiple computers accessing the same account allows cybercriminals to have each IP address send a lower number of emails, as low as one email per IP address in some cases. The longevity of the accounts that are accessed by more than one IP address suggests that the webmail service lacks effective countermeasures to prevent abuse of the service by such accounts. The community-based detection approach presented in this chapter allows an online service to promptly identify such accounts and thus prevent abuse and malicious activity.

Detecting Malicious Accounts Shared by Many IP Addresses. Can we use the fact that malicious accounts tend to be accessed by many IP addresses to flag these accounts as malicious? Figure 7.4 shows the Cumulative Distribution Function (CDF) of the number of IP addresses that accessed both malicious and legitimate accounts in **T**. As it can be seen, malicious accounts are more likely to be accessed by multiple IP addresses than legitimate accounts. Unfortunately, just looking at the number of IP addresses that accessed an account is not a strong enough indicator by itself. In fact, basing a detection system on the number of IP addresses that accessed an account would

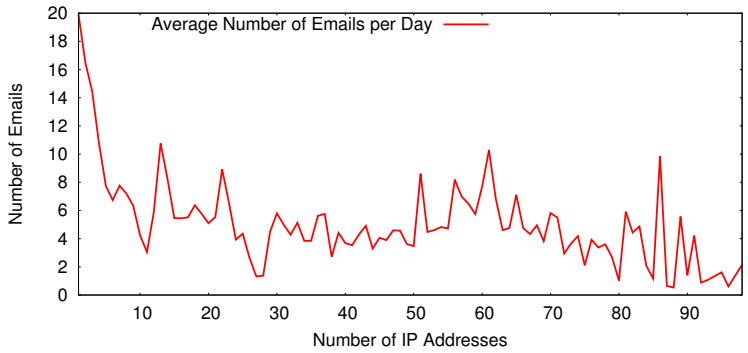


Figure 7.3: Number of spam emails sent per IP address that accessed a certain account. If many IP addresses access the same account, each of them can send a small number of emails and help keeping the malicious accounts under the radar from the webmail provider.

generate a number of false positives that is too high to be practical. For example, considering as malicious accounts that were accessed by two or more IP addresses in **T** would cause 77% of the total detections to be false positives (i.e., accounts that did not send any spam email). This makes sense, because many users access their webmail account from different devices, such as a mobile phone and a desktop computer. Even by looking at accounts accessed by a higher number of IP addresses does not solve the false positive problem: looking at accounts that were accessed by ten or more distinct IP addresses in **T** generates a 32% false positive rate; by increasing the number of required IP addresses the ratio of false positives decreases, but it remains well above the level considered acceptable in a production environment.

To overcome the false positive problem, we leverage another property of cybercriminal operations that use online services: cybercriminals can only count on a limited number of infected machines (bots). To be able to have each of their infected machines perform a small amount of interaction with the accounts under the control of the cybercriminal,

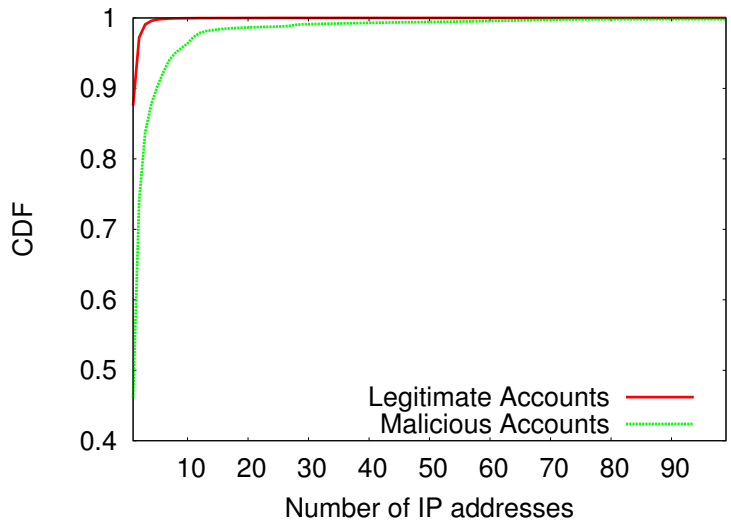


Figure 7.4: Cumulative Distribution Function (CDF) of the number of IP addresses that accessed benign accounts in **T** and malicious accounts in **L**. As it can be seen, malicious accounts are more likely to be accessed by multiple IP addresses than legitimate ones.

attackers have to make their bots connect to different accounts over time. We can think of a set of accounts that are accessed by the same set of bots as a *community*. In the following, we present EVILCOHORT, a system that detects communities of accounts that are accessed by a common set of IP addresses. We show that looking at these communities of accounts allows us to detect most of the malicious accounts that are accessed by multiple IP addresses, while generating a false positive rate that is orders of magnitude lower than just looking at accounts in isolation. In Section 7.5.1 we compare the two methods in details, and show that EVILCOHORT outperforms the single account method.

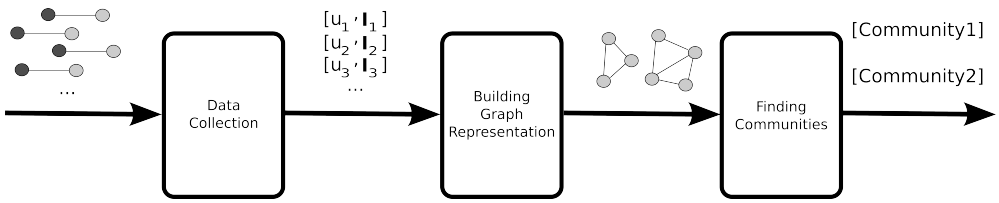


Figure 7.5: Overview of EVILCOHORT. The darker circles represent IP addresses, while the lighter circles represent online accounts.

7.3 EVILCOHORT: Overview

To operate, EVILCOHORT needs *account interaction events* as input. Users create their own accounts and connect to online services to perform a number of actions. Depending on the service, such actions can span from sending messages to the user's friends and colleagues, to performing friend requests, to browsing pictures, to updating the user's profile. Accounts allow the online service to attribute any activity performed to a specific user, in a more precise way than source IP addresses do. For instance, it is possible to correctly attribute the activity of a certain user regardless of the place she is connecting from (her home computer, her office, or her mobile phone). We define a user interaction with an online service as a tuple

$$\mathbf{A} =< H, U, T >,$$

where H is the host that the user is connecting from (identified by an IP address), U is her user ID on the online service, and T is a timestamp.

Approach Overview. EVILCOHORT works in three phases. First, it collects interaction events from the monitored online service, and builds a list of IP addresses that

accessed each of the observed accounts. Then, it builds a *graph representation* that represents which accounts have been accessed by the same set of IP addresses. As a third step, EVILCOHORT performs clustering on the combined access graph representation to find communities of online accounts. An overview of the different phases in EVILCOHORT's operation can be found in Figure 7.5. A last, optional step consists of analyzing the discovered communities, and applying a set of heuristics that can identify likely-legitimate communities and avoid false positives. In the remainder of this section, we provide more details about the steps involved in identifying communities.

7.3.1 Data Collection

In the first phase of operation, EVILCOHORT collects interaction events on an online service for an observation period (a day in our implementation). For each account that was active during the observation period, EVILCOHORT keeps an *access list* **I**, which contains the set of IP addresses that accessed that particular online account. At the end of this phase, EVILCOHORT returns a set $\mathbf{S}_A$, where each element is a tuple containing the user ID that identifies an account u and the access list **I** for that account.

7.3.2 Building the Graph Representation

We expect cybercriminals to have their bots connecting to multiple accounts under their control, because they have control of a limited number of bots, and want to achieve the effectiveness illustrated in Section 7.2. For this reason, we represent the relation between online accounts and IP addresses as a weighted graph. More precisely, we

define the *graph representation* of the set of accounts **U** as

$$\mathbf{R} =< \mathbf{V}, \mathbf{E} >,$$

where each element in the set of vertices **V** is one of the accounts in **U**, and the set of edges **E** is weighted as follows: for each pair of accounts u_1, $u_2 \in \mathbf{V}$, the edge connecting them has a weight equal to the number of IP addresses that u_1 and u_2 share. If the accounts u_1 and u_2 do not share any IP address there is no edge between them.

As we showed in Section 7.2, a consistent number of legitimate accounts are accessed by more than one IP address. To allow us to only detect communities of accounts that share a higher number of IP addresses, we introduce a threshold s, and consider as inputs for this phase only those accounts whose access lists **I** contain a number of IP addresses equal or greater than s. Since the number of IP addresses that legitimate accounts share is low, communities of accounts sharing many IP addresses are very likely to be suspicious. We investigate the possible choices for the threshold s in Section 7.5.1.

EVILCOHORT builds the graph representation as follows:

1. For each element $\mathbf{S}_A$, if the number of IP addresses in the access list **I** is higher than s, we create a node in the graph **R**, labeled with the user ID u.

2. For each pair of accounts u_1 and u_2, we calculate the intersection of the IP addresses that accessed them, by using the access lists $\mathbf{I}_{u_1}$ and $\mathbf{I}_{u_2}$. If the intersection is not empty, we create an edge in **R** between the nodes labeled as u_1 and u_2. The weight of the edge is the size of the IP address intersection.

The graph **R** is then passed to the next phase of our approach, which finds communities of online accounts that are accessed by a common set of IP addresses.

7.3.3 Finding Communities

After obtaining the graph representation **R**, we identify communities of accounts in it. To this end, we use the "*Louvain Method*" [36]. This clustering method leverages an iterative algorithm based on modularity optimization, and is particularly well-suited to operate on sparse graphs, as most graphs obtained from "real life" situations are [58]. In their paper, Blondel et al. [36] show that their method outperforms several community-detection algorithms based on heuristics.

The Louvain method operates in two phases, which are iteratively repeated until convergence is reached. At the beginning, each vertex in **R** is assigned to its own community of size one. Each iteration of the algorithm proceeds as follows:

1. For each account u_1 in **U**, we consider each of its neighbors u_2, and we calculate a gain value g that represents the effect that we would have by removing u_1 from its community and adding it to u_2's community. We explain how we calculate g later in this section. If any of the gain values g is positive, we move u_1 to the community of the account that returned the highest gain.

2. We rebuild the graph **R**, whose nodes are now the communities built during phase 1. Each edge between two communities c_1 and c_2 is weighted with the number of IP addresses that are shared between the two communities.

The algorithm repeats these two phases until convergence. Blondel et al. [36] describe how the gain value g is calculated in detail. In a nutshell, the gain obtained in moving an account i to a community C is

$$g_{in} = [\frac{\sum_{in} + k_{i,in}}{2m} - (\frac{\sum_{tot} + k_i}{2m})^2] - [\frac{\sum_{in}}{2m} - (\frac{\sum_{tot}}{2m})^2 - (\frac{k_i}{2m})^2],$$

where $\sum_{in}$ is the sum of the weights of the edges between the accounts in C, $\sum_{tot}$ is the sum of the weights of the edges incident to the accounts in C, k_i is the sum of the weights of the edges incident to i, $k_{i,in}$ is the sum of the weights of the edges that connect i to the accounts in C, and m is the number of edges in **R**. Blondel et al. show how a similar weight is calculated for the gain obtained by removing an account i from its community (g_{out}) [36]. If the sum of the two gains $g = g_{in} + g_{out}$ is positive, the account i gets added to the community C.

7.3.4 Postprocessing Step

The postprocessing phase takes the set of communities detected by EVILCOHORT and performs additional analysis to understand the behavior of these communities and extract some characteristic traits. Postprocessing consists of optional filters that can be applied to filter out potential false positives such as users behind a *Network Address Translation (NAT)* device that could have been wrongly attributed to a malicious community. Each filter is independent and can integrate additional information that is not necessarily part of the *account interaction events* or the *graph representation*, such as HTTP user agents and time patterns. In the following, we introduce filters that can be used to study the detected communities and reduce false positives even further.

User agent correlation filter

This filter relies on the assumption that a typical user connects to her account from a limited set of devices. A typical user would access, for example, her account from home using her personal browser, then from work using the browser recommended by the company policy, and finally from her phone using her mobile browser. In other words, we expect to have a one-to-one relationship between the sources of the activity and the agents used to perform the activity. When online services are accessed via the web, the agent used to perform the activity can be identified by the HTTP user agent field.

On the opposite side, in malicious communities, the activity is no longer generated by humans behind a browser but often generated by autonomous programs such as bots or programs used to administer multiple accounts at once. These programs can be designed to use either hard-coded user agent strings, or, as we observed in recent malware, a permutation of user agent strings. These strings are often chosen in a way to masquerade a legitimate browser. However, hard coding or permuting user agent strings results in multiple sources relating to a single string, or, conversely, a single source relating to multiple strings.

To measure the correlation between the sources and the user agents we compute the following ratio:

$$log(c) = log\left(\frac{\text{number of user agents}}{\text{number of IP addresses}}\right) \tag{7.1}$$

For a typical benign user, the correlation is very strong because there is a one-to-one relationship between source and user agent: each source is associated to a different

user agent, meaning that $log(c)$ tends towards 0. For malicious communities, where the relationship becomes one-to-n, negative values will be observed in case of hard-coded user agent strings, and positive values in case of permutations of the user agent strings.

Note that we exclude from the computation user agent strings coming from mobile phones or tablets because these mobile devices can be connected to any network, meaning that no correlation can be expected in this case.

Event-based time series filter

This filter introduces a different representation of *account interaction events* over time. Time series represent the frequency of events per time period. An example of this can be found in Figure 7.9.

This filter relies on the assumption that the shape of the time series for legitimate communities and malicious communities are fundamentally different. The techniques used by the time series filter are established in the research community, and were presented in a paper by Jacob et al. [74].

For legitimate users, we expect the shape of the time series to exhibit some daily patterns due to the night and day cycles, as well as some long term variations with the passing weeks. On the opposite, malicious communities will show either a high degree of stability and regularity, as in the case of botnets using online services as their command and control channels, or some irregular bursts, as in the case of spam campaigns being sent out.

Detection is based on automated classifiers working on statistical features characterizing the shape of the time series. For technical details, the reader is invited to refer to the original paper [74]. Note that this filter can only be applied on cases where the amount of events is sufficiently large to be statistically meaningful for the analysis tools that the technique relies on.

IP addresses and accounts usage filter

This filter, similarly to the previous one, relies on time analysis. The main difference lies in the fact that events are no longer aggregated for the full community but IP addresses and accounts being accessed are represented separately over time.

The IP addresses usage graph, is generated in the following way: time is represented on the x-axis and each unique IP address is represented by a separate entry on the y-axis of the graph. Events are then plotted as points in the graph using this set of coordinates. The account usage graph is generated in a similar way, with unique accounts instead on the y-axis. An example of this can be found in Figure 7.10. The goal of such representations is to discover some potential synchronization across IP addresses or accounts.

This filter relies on the assumption that events generated by users of a legitimate community are mostly asynchronous if not unrelated. On the opposite side, malicious communities tend to exhibit a high degree of synchronization in their events both across IP addresses, and across accounts. For example, multiple accounts might be accessed synchronously at the beginning of a spam campaign.

Using this type of representation, any suspicious alignment in the events recorded for different IP addresses or different accounts can easily be identified on these separate graphs. These alignments reveal some strong synchronization across users, which is highly suspicious.

7.4 Description of the Datasets

In Section 7.2 we analyzed **T**, a labeled dataset of email-sending events on an webmail provider. However, since EVILCOHORT only takes into account the mapping between IP address and online account of an event on an online service, it can operate on any online service that allows users to create accounts. Such services include web-based email services, online social networks, blogs, forums, and many others. In addition, EVILCOHORT can operate on activities of different types, such as login events, message postings, message shares, etc. To show the versatility of our approach, we evaluated it on multiple datasets of activities on five different online services. The first dataset is composed of email sending events logged by a large webmail service. The second dataset is composed of login events logged on four different online social networks. In the following, we describe these datasets in more detail.

7.4.1 Webmail Activity Dataset

Our first dataset is composed of email-sending events logged by a large webmail provider. Every time an email is sent, an activity is logged. We call this dataset $\mathbf{D}_1$. Note that the

email-sending events in this dataset are generated by accounts on the webmail service, which send emails to other accounts on the same service, or to the outside world.

The dataset $\mathbf{D}_1$ contains a random sample of the events logged over a five-month period by the webmail provider. In total, this dataset contains 1.2 billion email-sending events, generated by an average of 25 million accounts per day. This data was collected according to the webmail provider's terms of service, and was only accessed on their premises by a company's employee. In addition to the activity events, the webmail provider logged whether the email was flagged as spam by their anti-spam systems. We used a subset of $\mathbf{D}_1$, which we called **T**, to study the properties of legitimate and malicious accounts on a webmail service (see Section 7.2). We will also use **T** as ground truth for our further experiments.

7.4.2 Online Social Network Login Dataset

Online Social Network	**OSN**$_1$	**OSN**$_2$	**OSN**$_3$	**OSN**$_4$
Number of login events	14,077,316	311,144	83,128	42,655
Number of unique Accounts	6,491,452	16,056	25,090	21,066
Number of unique IPs	6,263,419	17,915	11,736	4,725
Average of daily events (week day)	2,067,486	51,832	11,897	6,601
Average of daily events (week end)	1,848,213	32,966	11,726	4,995
Percentage of account singletons (daily)	74.6%	40.0%	51.7%	72.2%
Percentage of account singletons (weekly)	65.4%	29.8%	51.5%	71.8%

Table 7.1: Statistics of activity events on our online social network login dataset.

Our second dataset is composed of login events on online social networks, spanning a period of 8 days. We call this dataset $\mathbf{D}_2$. The dataset contains login activities that happened on four different online social networks over a period of eight weeks. We

obtained the dataset $\mathbf{D}_2$ from a security company. For each activity event, the dataset contained additional information such as the user agent of the web browser performing the login and the HTTP headers of the response. Sensitive information such as the user ID and the IP address were anonymized. Note that this did not affect our community detection algorithm at all.

Statistics on the number of login events for each social network can be found in Table 7.1. These statistics reflect the size and activity observed on these networks, ranging from tens of thousands up to 14 million login events. One interesting observation is the high percentage of account singletons on a daily basis, that is to say the percentage of users connecting at most once a day. On a weekly basis, the percentage tends to drop but remain surprisingly high. These users are probably legitimate users that are not very active on the social network.

7.5 Evaluation

In this section, we analyze how EVILCOHORT performs in the real world. We first validate our approach by using the labeled dataset $\mathbf{T}$ of malicious and legitimate email-sending events. We then select a suitable threshold that allows us to have a small number of false positives. Finally, we run EVILCOHORT on multiple real-world datasets, and we analyze the communities of malicious accounts that we detected.

Value of s	# of accounts	# of communities	Coverage on L	Add. Detections	FP communities	FP accounts
2	135,602	3,133	94,874 (58%)	40,728 (23.8%)	1,327 (42%)	12,350 (9.1%)
5	77,910	1,291	51,868 (30.4%)	26,042 (15.2%)	580 (44.9%)	2,337 (3%)
10	25,490	116	16,626 (9.7%)	8,864 (7.7%)	48 (41.3%)	433 (1.7%)
65	1,331	6	1,247 (0.7%)	84 (0.04%)	0	0

Table 7.2: Summary of the results reported by EVILCOHORT for different values of the threshold s.

7.5.1 Threshold Selection

As with every detection system, EVILCOHORT has to make a trade-off between false negatives and false positives. As we mentioned in Section 7.3.2, we can adjust the value of the threshold s to influence the quality of EVILCOHORT's results. In particular, increasing the value of s decreases the number of false positives of our system, but also reduces the number of accounts that can be detected. In this section we run EVILCOHORT on the dataset **T** and analyze the quality of its results. The goal is to identify a suitable value of s for running EVILCOHORT in the wild. To this end, we take advantage of the set of labeled malicious accounts **L** that we introduced in Section 7.2.

The first element that we use to evaluate the effectiveness of EVILCOHORT is the fraction of accounts in **L** that our system is able to detect. We call this fraction the *coverage* of EVILCOHORT with respect to **L**. Ideally, we want EVILCOHORT to detect a large fraction of our labeled malicious accounts. Unfortunately, increasing the value of s decreases the number of accounts that EVILCOHORT can possibly detect, because it discards all accounts that have been accessed by less than s IP addresses. The coverage provides us an estimate of the false negatives that EVILCOHORT would report if it was run in the wild.

As a second element of effectiveness, we look at the set of accounts that EVILCOHORT detects as malicious in **T**, but that were missed by the anti-spam systems deployed by the webmail provider. These are malicious accounts *not* in **L**. We refer to this number as *additional detections*. This value gives us an estimate on the overall effectiveness of EVILCOHORT. Ideally, we want this number to be high, so that if EVILCOHORT were to be deployed in conjunction with the defenses that are already in place on the online service, it would increase the number of malicious accounts that can be detected and blocked.

The third element that we consider is the *confidence* that the communities detected by EVILCOHORT are indeed malicious. To this end, we look at the fraction of accounts in **L** that are present in each detected community. We consider a community as malicious (i.e., a true positive) if at least 10% of the accounts belonging to it are part of our labeled dataset of malicious accounts. Otherwise, we consider it as a false positive of EVILCOHORT. We empirically found that this fraction gives us a good confidence that these communities are indeed malicious. By incresing the fraction of accounts needed to asses false positives, results do not change significantly.

Table 7.2 provides a summary of the results that we obtained when running EVILCOHORT on **T**, based on different values of the threshold s. As one can see, the fraction of accounts in **L** that our system detects decreases quickly as we increase s. With a threshold of 2, EVILCOHORT only detects 58% of the labeled accounts, and this fraction decreases to 30% if we set s to 5. With a threshold of 10 the fraction of accounts in **L** that are covered is only 10%. Once we reach higher thresholds, such as 65, the fraction of detected accounts that are part of **L** becomes very small. The additional

detections performed by EVILCOHORT over the webmail provider's detection system also decrease as we increase s. With a threshold of 2 we detect 23% malicious accounts that existing approaches miss. This number decreases to 15% with a value of 5 for s; a threshold of 10 still ensures 7.7% additional detections.

False positives decrease rapidly as we increase s as well. Setting s to 2 results in 9% false positives, while a threshold of 5 already decreases false positives to 3%. A threshold of 10 reduces false positives to 1.7%. By setting s to 65, EVILCOHORT does not mistakenly flag any legitimate account as malicious. Unfortunately, the number of detections performed at this threshold is quite low.

Given the results reported in this section, we decided to use 10 as a value of s for our experiments. At this threshold false positives are low (1.7%), but the system is still able to significantly improve the detections performed by the existing countermeasures deployed by the webmail provider, and detects 116 communities as malicious.

It is interesting to notice that although the number of accounts misclassified by EVILCOHORT is generally low, percentages are higher when looking at communities; For example, with a threshold of 10, 40% of the detected communities are considered to be false positive accounts. Interestingly, however, the size of true positive and false positive communities varies consistently: false positive communities are composed of 9 accounts on average, while malicious ones are composed by 370 or more accounts. This indicates that filtering on the number of accounts in a community could be an effective filter to further reduce false positives.

Comparison between EVILCOHORT and the single account method. As we dis-

cussed in Section 7.2, EVILCOHORT outperforms detection by looking at single accounts accessed by a high number of IP addresses by orders of magnitude. At a threshold of 10, where EVILCOHORT reports a false positive rate of 1.7%, the single-account method has a false positive rate of 32%. Even by dramatically increasing the threshold, the number of false positives of the single-account method remains high. At a threshold of 65, at which EVILCOHORT reports no wrong detections, the single-account method has a false positive rate of 1.2%. Even at a threshold of 100, the single-account method has a small number of false positives.

The last question to answer is whether accounts that are accessed by a high number of IP addresses do form communities, in other words whether EVILCOHORT is able to detect most malicious accounts that were accessed by a number of IP addresses s. To answer this question, we looked at single accounts accessed by a number of IP addresses n (from one to 100), and labeled them as malicious or benign in the same way we labelled the communities in the previous experiment. We then proceeded as follows: for each value of n, we considered the single-account method to have perfect recall (i.e., no false negatives). We then looked at how many of the accounts detected by this method would have formed communities, and therefore be detected by EVILCOHORT. The fraction of malicious accounts that form communities is generally very high. With a threshold of 10, EVILCOHORT detected 93% of the malicious accounts detected by the single-account method. With a threshold of 20 this fraction becomes 95%, while with a threshold of 50 it becomes 98%. We conclude that the vast majority of accounts accessed by a high number of IP addresses form communities, and that therefore EVILCOHORT is a suitable alternative to the single-account method for what concerns false negatives, and it reduces false positives by orders of magnitude

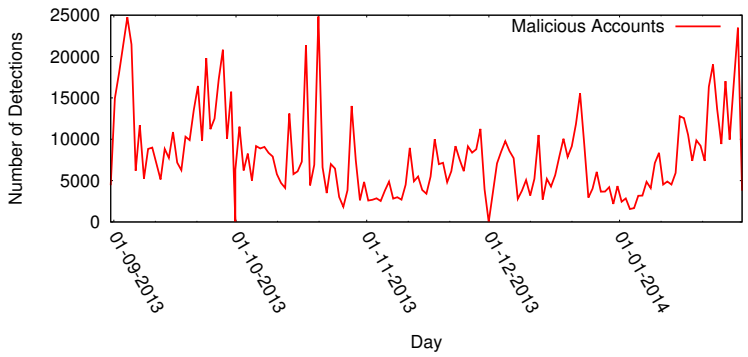


Figure 7.6: Number of malicious accounts detected per day by EVILCOHORT on the dataset $\mathbf{D}_1$.

compared to the single account method.

7.5.2 Detection in the Wild

We applied EVILCOHORT to the datasets $\mathbf{D}_1$ and $\mathbf{D}_2$. In the following, we show that EVILCOHORT is able to detect a large number of malicious online service accounts, regardless of the type of online service that it is ran on.

Detection on the Webmail Activity Dataset

The dataset $\mathbf{D}_1$ is composed of email-sending activities logged on a large webmail provider. Over a period of 5 months, EVILCOHORT was able to detect 1,217,830 accounts as malicious. Note that, in this context, a malicious account is an account that is misused to send spam emails. Of these accounts 1,058,830 were detected as malicious only once, while 60,708 accounts were detected as malicious multiple times during

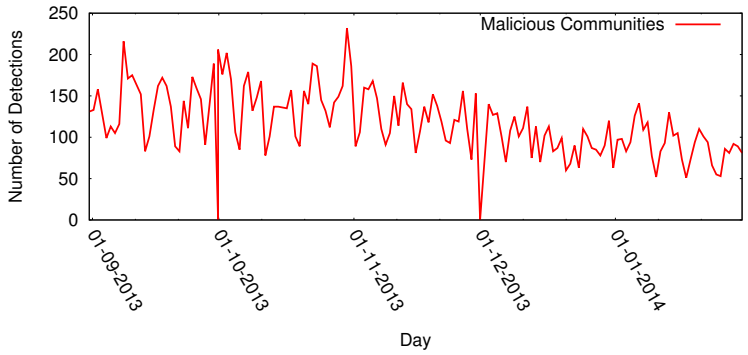


Figure 7.7: Number of malicious communities of accounts detected per day by EVILCOHORT on the dataset $\mathbf{D}_1$.

our observation period (recall that we use observation periods of one day). In total, EVILCOHORT detected 17,803 malicious communities.

We then wanted to understand how much EVILCOHORT is able to grow the knowledge of malicious accounts compared to the defenses already put in place by the webmail provider. Of the accounts that we detected, 502,159 of them never had an email flagged as spam by the email provider. This accounts for 41% of the total. To assess how many of the detections performed by EVILCOHORT were potential false positives, we performed the same analysis that we described in Section 7.5.1. We considered an account as a false positive if less than 10% of the accounts in a community were detected as malicious. In total, we found 1.9% of the detected accounts to be potential false positives. This is in line with the validation results from Section 7.5.1, in which we reported 1.7% false positives by using the same threshold. 94.6% of the total accounts detected as malicious by the webmail provider at this threshold formed communities, and were therefore detected by EVILCOHORT. This shows that malicious accounts accessed by a large number of IP addresses are typically accessed by botnets, and confirm

Day	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
OSN$_1$	24	30	6	4	4	4	5	2
OSN$_2$	1	0	0	0	0	0	0	0
OSN$_3$	0	0	0	0	1	1	1	0
OSN$_4$	0	0	0	0	0	0	0	0

Table 7.3: Number of malicious communities detected per day by EVILCOHORT on the dataset $\mathbf{D}_2$.

Social Network	**OSN$_1$**	**OSN$_2$**	**OSN$_3$**	**OSN$_4$**
Accounts (Avg)	3,662	2	2	0
Accounts (Med)	13	2	2	0
Accounts (Max)	66,764	2	2	0
IPs (Avg)	2,381	14	10	0
IPs (Med)	19	14	10	0
IPs (Max)	3,9884	14	10	0

Table 7.4: Size of the malicious communities detected by EVILCOHORT on the dataset $\mathbf{D}_2$. Numbers (Average, Median and Maximum) are expressed per community.

the usefulness of our approach.

Detection on the Social Networks Dataset

The dataset $\mathbf{D}_2$ is composed of login events from online social networks. In the following, we applied EVILCOHORT to this second dataset to prove the viability of the approach for different types of services. For these experiments, no ground truth was available. For this reason, we used the same threshold as selected in Section 7.5.1. To confirm that the accounts belonging to these communities were indeed malicious, we performed a postprocessing analysis as we described in Section 7.3.4. We discuss the results of these experiments in Section 7.5.3.

Over the 8 days, EVILCOHORT was able to detect a total of 83 communities, which

represents a total of 111,647 unique accounts. The number of detected communities and the size of these communities evolve daily as it can be observed in Table 7.3 and Table 7.4. Unsurprisingly, these numbers heavily depend on the size of the social network. OSN_1 is by far the largest network; consequently, this is where we observed the highest number of communities, as well as the largest communities. Interestingly, we observe an important drop in the number of communities on the third day meaning that the accounts might have been taken down by the network after they detected some malicious activities. The remaining communities tend to be of smaller size. In OSN_2 and OSN_3 we only detect isolated communities of very small size: two accounts for about ten different IP addresses. The activity for OSN_4 was too little for us to detect any interesting community with the selected threshold.

To understand the evolution of the communities, we studied their similarity over time. A community remains stable over time if it is found similar to a community detected the day before. We declare two communities as similar if they share more than 50% of their accounts. In OSN_1, one of the largest communities, with more than 66,000 accounts, was stable for over five days with a similarity ranging between 53% to 85%. Two communities of smaller size, with about 10,000 accounts each, were found stable over the two first days but they disappeared on the third day as previously observed. The community detected in OSN_3 is only made up of two accounts and is stable over three days before being brought down.

Beside the stability over time, we studied the possible overlap of communities across social networks. To study this overlap, we focused on the IP addresses hosting some malicious community activity. Overall, only OSN_1, OSN_2 and OSN_3 share about 2,000

source IP addresses. Out of these, we found no IP addresses belonging to one of the detected communities. In other words, we could not obtain any proof of communities acting across multiple online social networks.

Since no ground truth was available for the dataset $\mathbf{D}_2$, we cannot be sure that the detected communities are actually malicious. For this reason, we analyzed the detected communities using the post-processing techniques described in Section 7.3.4. As expected, the vast majority of the communities of accounts detected by EVILCOHORT shows very different characteristics than legitimate communities, giving us confidence that EVILCOHORT is indeed effective in identifying communities of malicious accounts. In the following section we describe these experiments in detail.

7.5.3 Result Analysis

Given the lack of a labeled set for the dataset $\mathbf{D}_2$, we ran the postprocessing filters described in Section 7.3.4 on it. The results show that most communities of accounts detected by EVILCOHORT show very different characteristics than legitimate communities, and are therefore very likely to be malicious. In the following, we describe our experiments in detail. We did not run the postprocessing filters on $\mathbf{D}_1$ because not all the information needed for these filters was available to us.

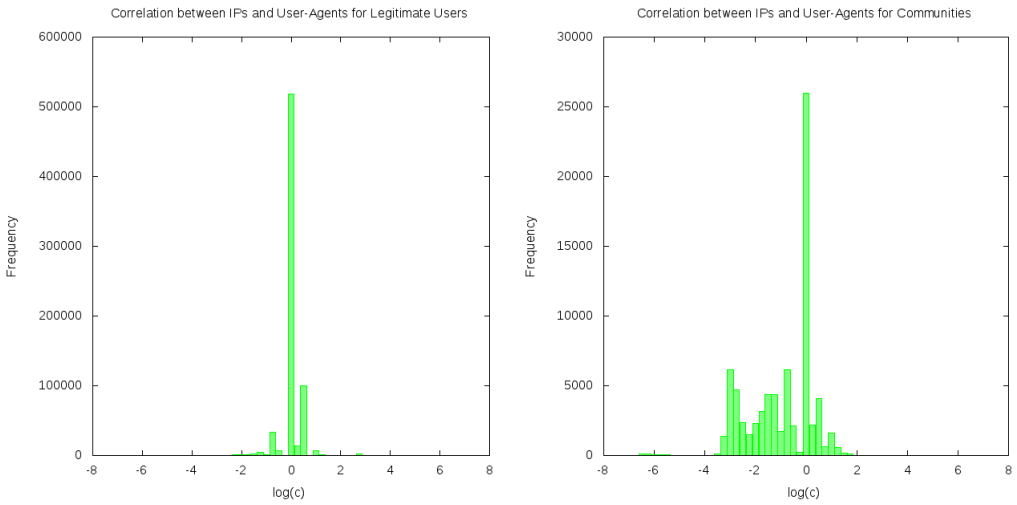


Figure 7.8: Correlation between user agents and IPs: legitimate accounts (left) and malicious accounts (right).

User Agent Correlation Filter

Information about user agents was only available for OSN_1 and OSN_2 in $\mathbf{D}_2$. Consequently, we excluded OSN_3 and OSN_4 from this study. We also excluded all the account singletons identified in Section 7.4.2 because the notion of ratio then becomes meaningless.

To verify the soundness of filtering communities based on user agents as described in Section 7.3.4, we plotted in Figure 7.8 the separated distributions of the correlation between user agents and IP addresses: one for legitimate accounts and one for accounts being part of detected communities. As it can be seen, the distribution is shifted for malicious communities and no longer aligned on the origin. For legitimate accounts, the average of $log(c)$ was 0.08, which is close to zero as expected, with a standard deviation of 0.43. For malicious communities, the average shifts to -0.85 with a standard

deviation of 1.24.

Using this filter, we evaluated the quality of the detection in OSN_1 and OSN_2, and tried to extract interesting characteristics of malicious communities. We observed an interesting phenomenon in communities that we considered true positives according to the filter. These communities often exhibit a high degree of similarity in terms of user agents across multiple accounts, but also across multiple communities. These results are detailed for both network in Table 7.5. Communities sharing a common user agent are likely to be part of a bigger campaign of malicious activity.

Network	Com	Acc	$log(c)$	UA Sim	UA string
OSN_1	1	11587	-1.5	100%	Mozilla/5.0 (Windows; U; [...]) Gecko/20070725 Firefox/2.0.0.6
OSN_1	22	268	-2.0	98%	Mozilla/5.0 (Windows NT 6.1; [...]) Gecko/20100101 Firefox/11.0
OSN_2	1	2	-2.4	100%	Mozilla/5.0 (Windows NT 6.1; rv:7.0.1) Gecko/20100101 Firefox/7.0.1

Table 7.5: Correlating malicious communities based on the User-Agent. Com - Number of Communities, Acc - Number of Accounts, $log(c)$ - Average correlation, UA Sim - User-Agent Similarity, UA string - User-Agent String.

If we look closely at the OSN_1 communities from the third entry of Table 7.5, we can see that the user agent similarity is not perfect. We actually observed some sporadic accounts with unrelated user agents and a correlation $log(c)$ close to 0. These accounts were showing additional user agents meaning that these accounts might actually have been compromised, but still accessed by their legitimate owners in addition to cyber-criminals. Nonetheless, the suspiciousness of these communities is confirmed by the fact that these are the communities we saw being brought down two days after the start of our dataset as described in the detection results of Section 7.5.2.

While we evaluated the quality of the detected communities, we also looked for potential false positives. We actually found two communities detected in OSN_1, one of

5,720 accounts and one of seven accounts, that were likely false positives according to the filter. For all accounts coming from these two communities, the correlation index $log(c)$ was very close to zero. The number of accounts in these communities might seem significant, but further postprocessing will show that these accounts are not all false positives. A majority of these accounts actually fall in the category previously observed of compromised accounts where both legitimate and malicious activity are simultaneously observed. This is particularly true for the largest community where we will actually observe some suspicious behaviors in coming experiments. For OSN_2, the filter detected no potential false positive.

Event-based Time Series Filter

Since time series become only significant if the amount of data is sufficiently large to make any measure statistically meaningful. For this reason, we evaluated the time series filter introduced in Section 7.3.4 over OSN_1. Unfortunately, the volume of login events observed for OSN_2, OSN_3 and OSN_4, as well as the size of the detected communities made this approach impracticable for these networks.

The assumption behind time series filtering is that part of the events observed in malicious communities are triggered by automated entities, eventually resulting in a distinct shape of activity from communities of legitimate users where events are triggered by humans [74]. To verify this assumption, we plotted the time series associated to the activity of the biggest malicious communities detected in OSN_1. The experiments showed that the time series generated for malicious communities differ fundamentally in shape from regular user activity, even when users are grouped behind a NAT.

Concrete examples are plotted in Figure 7.9. The leftmost time series represents the activity of all users from OSN_1 over 8 days. The reader can clearly see the daily patterns in the activity. The middle left time series represents the activity we observed for an isolated IP where multiple accounts were being accessed behind a NAT. One can see that the time series remains very similar in shape and preserves the daily patterns. The middle right time series represents the activity generated by the largest malicious community detected in OSN_1 (first entry in Table 7.5). As it can be seen, there are fundamental differences: disappearance of the daily patterns, higher stability on the long term. The rightmost time series is representative of most of the time series obtained for smaller communities of OSN_1: the volume of events remains low but one can clearly observe regular bursts of activity. This bursty shape is also observed for the potential false positive of 5,720 accounts detected by the user agent filter. The only exception is the community of seven accounts detected as false positive by the user account filter.

IP Addresses and Accounts Usage Filter

Time series being restricted to large amounts of data, another possible representation of the activity over time is to plot the usage graphs for IP addresses and accounts as detailed in Section 7.3.4. For malicious communities, the usage graphs will exhibit suspicious patterns betraying some synchronization across accounts and IP addresses. For reference, Figure 7.10 presents the usage graphs for the exact same NAT as in Figure 7.9. One can see clearly in the account usage graph the daily interruptions over night time as well as the randomness of the events during day time.

If we look at malicious communities as plotted in Figure 7.12, one can see some sus-

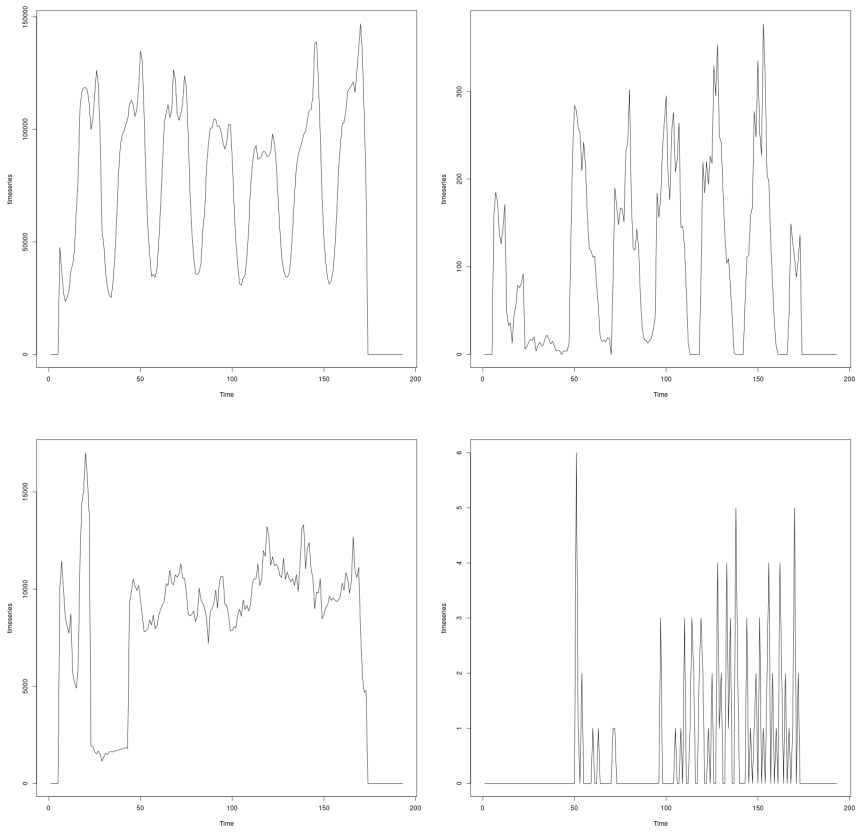


Figure 7.9: Time series plotting login event over time: all accounts (leftmost), legitimate accounts behind a NAT (middle left) and malicious communities (middle right and rightmost).

picious vertical patterns appearing. Looking at the IP addresses usage graphs, IP addresses are active by synchronous groups. These groups shift everyday and IP addresses are rarely reused over time, probably as a protection to avoid detection. Looking at accounts usage graphs, accounts, on the other hand, remain stable over time but login events are also suspiciously synchronized. This phenomenon can be observed for all detected communities of OSN_1, OSN_2 and OSN_3 at the exception of the two potential false positives of OSN_1.

Looking more closely at the second example of Figure 7.12, some accounts and IP addresses have been clearly associated to the community even if they exhibit some benign activity. These accounts correspond to the examples from Table 7.5 where we observed some discrepancies in the user agent filtering. This observation confirms our suspicions that some accounts are probably compromised but still accessed by their legitimate owner.

In comparison, if we look back at the potential false positive of 5,720 accounts from OSN_1 discovered in Section 7.5.3, one can see in Figure 7.11 that the usage graphs are overall similar to the NAT usage graphs. However, looking more closely, one can observe multiple darker vertical patterns in the graphs that can be related to the patterns we observed in malicious communities. The mix of legitimate and malicious activities must have confused the user agent filter but the detection by EVILCOHORT in this case is confirmed by the two filters based on time analysis. On the other hand, the usage graphs for the smaller false positive of seven accounts from OSN_1 did not exhibit again any suspicious pattern.

7.6 Discussion

We showed that EVILCOHORT can be applied to very different online services and to any type of activity on such services. This versatility, together to the fact that it complements detections made by state-of-the-art systems makes EVILCOHORT a useful tool in the fight against malicious activity on online services. As any detection system, however, EVILCOHORT has some limitations. In this section, we describe our system's

limitation, as well as some possible directions for future work.

The main limitation of EVILCOHORT, as we already mentioned, is that it can only detect malicious accounts that are accessed by a large number of IP addresses. As we showed in Section 7.2, however, such accounts are more dangerous than the ones that are accessed by single IP addresses, and existing countermeasures are able to shut down this second type of accounts much quicker.

Another shortcoming is that the accounts used by cybercriminals are not necessarily fake accounts, but could be legitimate accounts that have been compromised. In the current implementation, EVILCOHORT cannot distinguish between the two types of accounts. Dealing with compromised accounts is more difficult, because the online service cannot just suspend them, but has to go through expensive password-reset operations. In the future, we could improve EVILCOHORT to detect the characteristics of the accounts in communities, to detect whether they are fake or compromised. We partially explored this direction in Section 7.5.3. Another way to do this would be to look at accounts that have an established history of legitimate activity, and suddenly start being part of a community.

Since EVILCOHORT can work, among the rest, on login events, it has the potential of detecting an account as malicious and blocking it before it performs any malicious activity. In the current version, the system works in batches, on time frames of one day. In the future, however, we plan to extend it to handle a stream of data, and operate in real time. This way, the system could continuously build account communities, and flag an account as malicious as soon as it joins a community.

In Section 7.5, we explored a variety of possible threshold at which EVILCOHORT could operate, and selected one that ensures a low rate of false positives. However, using a lower threshold could still ensure a small number of false positives, if used in conjunction with the postprocessing filters described in Section 7.3.4. As future work, we plan to explore different tradeoffs between the number of IP addresses accessing a community and the characteristics of such community.

7.7 Conclusions

We presented EVILCOHORT, a system that detects malicious accounts on online services by identifying communities of accounts that are accessed by a common set of computers. Our results show that the vast majority of the accounts that form such communities are used for malicious purposes. In the rare cases in which legitimate communities of accounts form, we show that such communities present behavioral characteristics that are very different than the ones of malicious communities. These differences can be used to perform more accurate detection. We ran EVILCOHORT on two real-world datasets, and detected more than a million malicious accounts.

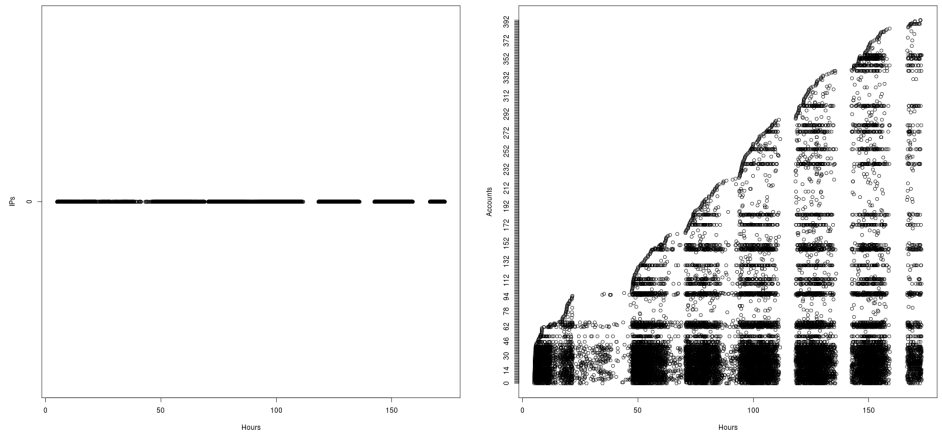
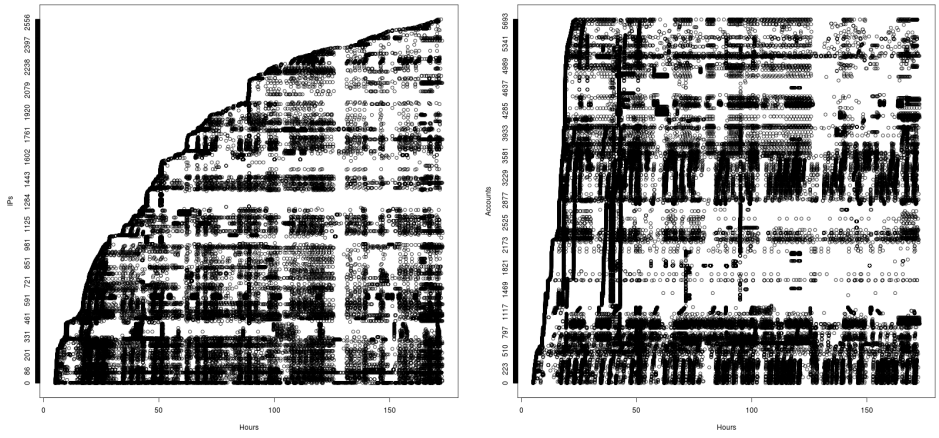


Figure 7.10: Activity of legitimate users behind a NAT: IP address usage (left) and account usage (right).



[t]

Figure 7.11: Activity of non malicious community: IP address usage (left) and account usage (right).

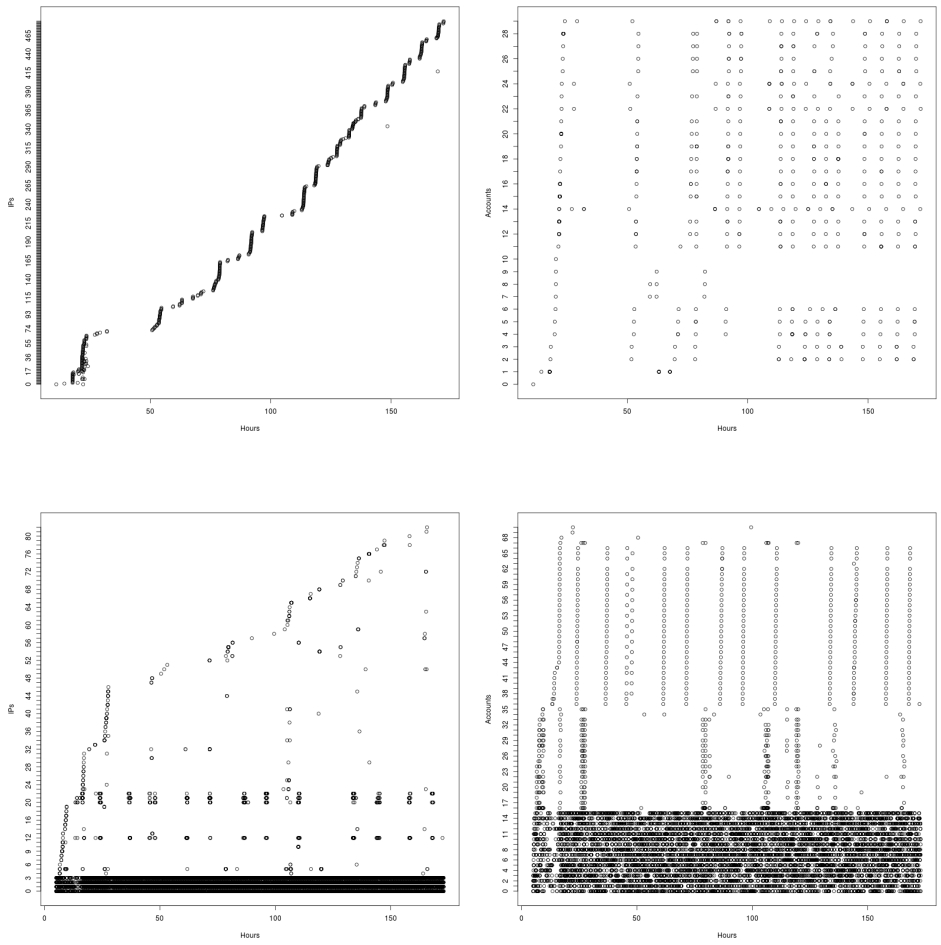


Figure 7.12: Activity of malicious communities: IP address usage (left) and accounts usage (right).

Chapter 8

Conclusions and Future Work

Cybercrime is currently a serious problem, and will remain a problem in the coming years. I described the three elements that are leveraged by cybercriminals to set up a large-scale malicious campaign targeting online services: infected computers, a command and control infrastructure, and online service accounts. I showed that, although a wealth of research has been conducted in detecting and disrupting such operations, the problem is far from being solved. The reason is that cybercriminals and researchers are involved in an arms race: every time a new detection technique is developed, miscreants come up with more advanced attack and management strategies that evade such technique.

In this dissertation, I showed that there are differences in the way in which regular users and cybercriminals use legitimate third party services. These differences originate from the different goal of these users: while legitimate users use online services for the pur-

poses they were designed for, cybercriminals exploit them for profit. I presented multiple systems that detect and block malicious activity by leveraging the differences in which infected machines and legitimate users use online services. Since these detection systems are based on elements that are important for the profitability of cybercriminal operations, they are difficult to evade, and raise the bar in the war against cybercrime.

I presented BOTMAGNIFIER, a system that learns the email-sending behavior of infected computers belonging to spamming botnets, and finds other computers that behave in the same way, effectively growing the knowledge of the population of bots belonging to that botnet. This system is a valuable aid to keep track of the infection population of large botnets, and improve the list of infected machines kept by blacklists, which notoriously have coverage problems.

As a second topic I focused on detecting accounts on online services that are misused by cybercriminals. I first presented one of the first studies on spam activity on online social networks, and introduced SPAMDETECTOR, a system that leverages the typical *modus operandi* of fake accounts controlled by cybercriminals on social networks to detect them. I then presented COMPA, the first system to detect legitimate social network accounts that have been compromised and are misused by cybercriminals. I showed that COMPA is effective in detecting both large-scale compromises, which target thousand of victims and spread spam or malware, and high-profile compromises, which target a singe account belonging to a news agency or a large company with the goal of spreading false information.

Finally, I focused on studying the relations between infected machines and accounts on online services. I showed that accounts that are accessed by many infected machines

(i.e., by botnets) are more dangerous for online services, because they manage both to stay active longer and to spread a higher quantity of malicious content. I presented EVILCOHORT, a system to detect communities of online accounts that are used by the same botnet. I showed that EVILCOHORT can be ran on any online service, and detect types of malicious activity that go beyond spreading malicious content.

In the future, I plan to keep studying Internet threats. As these threats become more sophisticated, the techniques required to fight them will need to be more advanced too. In particular, as attacks become more targeted and higher profile, it will not be possible to leverage their scale as I have done in this thesis to fight them. Some of the elements that we proposed, such as the behavioral modelling of COMPA, could however be adapted to fight these new threats. In my future work I plan to explore the use of such behavioral modelling technique to fight certain types of targeted attacks, such as spearphishing scams. On the social network side, I plan to combine the techniques presented in this dissertation with network analysis methods, to develop systems able to detect messages that are spreading anonymously.

Bibliography

[1] Attackers target yahoo mail accounts in coordinated effort to own users — threatpost. http://bit.ly/1nBgtqj.

[2] Compete site comparison. http://siteanalytics.compete.com/facebook.com+myspace.com+twitter.com/.

[3] Exposure. http://exposure.iseclab.org/.

[4] Facebook statistics. http://www.facebook.com/press/info.php?statistics.

[5] foursquare. http://foursquare.com.

[6] Google safebrowsing. http://code.google.com/apis/safebrowsing/.

[7] Honeypots. http://en.wikipedia.org/wiki/Honeypot_computing.

[8] Livejournal. http://www.livejournal.com.

[9] Nielsen. http://blog.nielsen.com.

[10] Oauth community site. http://oauth.net.

[11] Phishtank. http://www.phishtank.com.

[12] The recaptcha project. http://recaptcha.net/.

[13] Spamhaus dbl. http://www.spamhaus.org.

[14] Surbl. http://www.surbl.org.

[15] Tinyurl. http://tinyurl.com/.

[16] U.s. stocks tank briefly in wake of associated press twitter account hack. http://allthingsd.com/20130423/u-s-stocks-tank-briefly-in-wake-of-associated-press-twitter-account-hack/.

[17] Weka - data mining open source program. http://www.cs.waikato.ac.nz/ml/weka/.

[18] Wepawet. http://wepawet.iseclab.org.

[19] Youtube. http://www.youtube.com.

[20] Fox news's hacked twitter feed declares obama dead. http://www.guardian.co.uk/news/blog/2011/jul/04/fox-news-hacked-twitter-obama-dead, 2011.

[21] Renren. http://www.renren.com, 2011.

[22] Twitter finally released a "stalkers" app? no, it's a phishing scam. http://nakedsecurity.sophos.com/2011/08/12/twitter-

finally-released-a-stalkers-app-no-its-a-phishing-scam/, 2011.

[23] http://theonion.github.io/blog/2013/05/08/how-the-syrian-electronic-army-hacked-the-onion/, 2013.

[24] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. N-gram-based detection of new malicious code. In *Computer Software and Applications Conference (COMPSAC)*, 2004.

[25] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *ACM SIGCOMM Internet Measurement Conference (IMC)*, 2006.

[26] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a dynamic reputation system for dns. In *USENIX Security Symposium*, 2010.

[27] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou II, and D. Dagon. Detecting malware domains at the upper dns hierarchy. In *USENIX Security Symposium*, 2011.

[28] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *USENIX Security Symposium*, 2012.

[29] Apache Foundation. Spamassassin. http://spamassassin.apache.org.

[30] J. Baltazar, J. Costoya, and R. Flores. Koobface: The largest web 2.0 botnet explained. 2009.

[31] D. Balzarotti, M. Cova, C. Karlberger, E. Kirda, C. Kruegel, and G. Vigna. Efficient detection of split personalities in malware. In *Symposium on Network and Distributed System Security (NDSS)*, 2010.

[32] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida. Detecting Spammers on Twitter. In *Conference on Email and Anti-Spam (CEAS)*, 2010.

[33] F. Benevenuto, T. Rodrigues, V. Almeida, J. Almeida, and M. Gonçalves. Detecting spammers and content promoters in online video social networks. In *ACM SIGIR conference on Research and development in information retrieval*, 2009.

[34] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Symposium on Network and Distributed System Security (NDSS)*, 2011.

[35] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks. In *World Wide Web Conference (WWW)*, 2009.

[36] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[37] L. Breiman. Random forests. In *Machine Learning*, 2001.

[38] J. Caballero, C. Grier, C. Kreibich, and V. Paxson. Measuring pay-per-install: The commoditization of malware distribution. In *USENIX Security Symposium*, 2011.

[39] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-engineering. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[40] Z. Cai and C. Jermaine. The latent community model for detecting sybils in social networks. In *Symposium on Network and Distributed System Security (NDSS)*, 2012.

[41] Y. Cao, V. Yegneswaran, P. Possas, and Y. Chen. Pathcutter: Severing the self-propagation path of xss javascript worms in social web networks. In *Symposium on Network and Distributed System Security (NDSS)*, 2012.

[42] W. B. Cavnar and J. M. Trenkle. N-gram-based text categorization. In *Annual Symposium on Document Analysis and Information Retrieval (SDAIR)*, 1994.

[43] K. Chiang and L. Lloyd. A Case Study of the Rustock Rootkit and Spam Bot. In *USENIX Workshop on Hot Topics in Understanding Botnet*, 2007.

[44] C. Cho, J. Caballero, C. Grier, V. Paxson, and D. Song. Insights from the Inside: A View of Botnet Management from Infiltration. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2010.

[45] C. Y. Cho, E. C. R. Shin, D. Song, et al. Inference and analysis of formal models of botnet command and control protocols. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[46] M. Christodorescu and S. Jha. Testing malware detectors. In *ACM SIGSOFT Software Engineering Notes*, 2004.

[47] M. Christodorescu and S. Jha. Semantics-aware malware detection. Technical report, 2005.

[48] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. Technical report, DTIC Document, 2006.

[49] Z. Chu, S. Giannivecchio, H. Wang, and S. Jajodia. Who is tweeting on Twitter: human, bot, or cyborg? In *Annual Computer Security Applications Conference (ACSAC)*, 2010.

[50] Cisco Inc. Cisco IOS NetFlow. `https://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html`.

[51] E. Cooke, F. Jahanian, and D. McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2005.

[52] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *World Wide Web Conference (WWW)*, 2010.

[53] C. Curtsinger, B. Livshits, B. G. Zorn, and C. Seifert. Zozzle: Fast and precise in-browser javascript malware detection. In *USENIX Security Symposium*, 2011.

[54] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A taxonomy of botnet structures. In *Annual Computer Security Applications Conference (ACSAC)*, 2007.

[55] D. Dagon, C. C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *Symposium on Network and Distributed System Security (NDSS)*, 2006.

[56] H. Drucker, D. Wu, and V. N. Vapnik. Support vector machines for spam categorization. In *IEEE transactions on neural networks*, 1999.

[57] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna. COMPA: Detecting Compromised Accounts on Social Networks. In *Symposium on Network and Distributed System Security (NDSS)*, 2013.

[58] I. J. Farkas, I. Derényi, A.-L. Barabási, and T. Vicsek. Spectra of real-world graphs: Beyond the semicircle law. *Physical Review E*, 2001.

[59] F. C. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *European Symposium on Research in Computer Security (ESORICS)*, 2005.

[60] H. Gao, Y. Chen, K. Lee, D. Palsetia, and A. Choudhary. Towards online spam filtering in social networks. In *Symposium on Network and Distributed System Security (NDSS)*, 2012.

[61] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Zhao. Detecting and Characterizing Social Spam Campaigns. In *ACM SIGCOMM Internet Measurement Conference (IMC)*, 2010.

[62] C. Ghiossi. Explaining facebook's spam prevention systems. `http://blog.facebook.com/blog.php?post=403200567130`, 2010.

[63] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[64] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-independent Botnet Detection. In *USENIX Security Symposium*, 2008.

[65] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *USENIX Security Symposium*, 2007.

[66] G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *USENIX Security Symposium*, 2008.

[67] S. Hao, N. A. Syed, N. Feamster, A. G. Gray, and S. Krasser. Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine. In *USENIX Security Symposium*, 2009.

[68] M. Heiderich, T. Frosch, and T. Holz. Iceshield: detection and mitigation of malicious websites with a frozen dom. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2011.

[69] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and detecting fast-flux service networks. In *Symposium on Network and Distributed System Security (NDSS)*, 2008.

[70] X. Hu, M. Knysz, and K. G. Shin. Rb-seeker: Auto-detection of redirection botnets. In *Symposium on Network and Distributed System Security (NDSS)*, 2009.

[71] D. Y. Huang, H. Dharmdasani, S. Meiklejohn, V. Dave, C. Grier, D. McCoy, S. Savage, N. Weaver, A. C. Snoeren, and K. Levchenko. Botcoin: monetizing stolen cycles. In *Symposium on Network and Distributed System Security (NDSS)*, 2014.

[72] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar. Adversarial machine learning. In *ACM Workshop on Security and Artificial Intelligence*, 2011.

[73] J. Iedemska, G. Stringhini, R. Kemmerer, C. Kruegel, and G. Vigna. The Tricks of the Trade: What Makes Spam Campaigns Successful? In *Proceedings of the International Workshop on Cyber Crime (IWCC)*, 2014.

[74] G. Jacob, E. Kirda, C. Kruegel, and G. Vigna. Pubcrawl: protecting users and businesses from crawlers. In *USENIX Security Symposium*, 2012.

[75] T. Jagatic, N. Johnson, M. Jakobsson, and T. Jagatif. Social phishing. *Communications of the ACM*, 2007.

[76] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy. Studying Spamming Botnets Using Botlab. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[77] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. Voelker, V. Paxson, and S. Savage. Spamalytics: An empirical analysis of spam marketing conversion. In *ACM Conference on Computer and Communications Security (CCS)*, 2008.

[78] C. Kanich, N. Weaver, D. McCoy, T. Halvorson, C. Kreibich, K. Levchenko, V. Paxson, G. Voelker, and S. Savage. Show Me the Money: Characterizing Spam-advertised Revenue. *USENIX Security Symposium*, 2011.

[79] M. Khmartseva. Email Statistics Report. `http://www.radicati.com/wp/wp-content/uploads/2009/05/email-stats-report-exec-summary.pdf`, 2009.

[80] D. M. Kienzle and M. C. Elder. Recent worms: a survey and trends. In *Proceedings of the ACM workshop on Rapid malcode*, 2003.

[81] klout. `http://klout.com`.

[82] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-y. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *USENIX Security Symposium*, 2009.

[83] B. Krebs. Taking Stock of Rustock. `http://krebsonsecurity.com/2011/01/taking-stock-of-rustock/`, 2011.

[84] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. On the Spam Campaign Trail. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[85] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamcraft: An Inside Look at Spam Campaign Orchestration. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[86] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *World Wide Web Conference (WWW)*, 2010.

[87] K. Lee, J. Caverlee, and S. Webb. Uncovering social spammers: social honeypots + machine learning. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010.

[88] S. Lee and J. Kim. Warningbird: Detecting suspicious urls in twitter stream. In *Symposium on Network and Distributed System Security (NDSS)*, 2012.

[89] B. Leiba. DomainKeys Identified Mail (DKIM): Using digital signatures for domain verification. In *Conference on Email and Anti-Spam (CEAS)*, 2007.

[90] A. Lelli. Return from the Dead: Waledac/Storm Botnet Back on the Rise. `http://www.symantec.com/connect/blogs/return-dead-waledacstorm-botnet-back-rise`, 2011.

[91] M. Lindorfer, C. Kolbitsch, and P. M. Comparetti. Detecting environment-sensitive malware. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2011.

[92] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Conference on Email and Anti-Spam (CEAS)*, 2005.

[93] L. Lu, V. Yegneswaran, P. Porras, and W. Lee. Blade: an attack-agnostic approach for preventing drive-by malware infections. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[94] M86. Rustock, the king of spam. `http://www.m86security.com/labs/traceitem.asp?article=1362`, July 2010.

[95] MaxMind. GeoIP. `http://www.maxmind.com/app/ip-location`.

[96] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*. 2002.

[97] MessageLabs. MessageLabs Intelligence: 2010 Annual Security Report. `http://www.messagelabs.com/mlireport/MessageLabsIntelligence_2010_Annual_Report_FINAL.pdf`, 2010.

[98] T. Meyer and B. Whateley. SpamBayes: Effective open-source, Bayesian based, email classification system. In *Conference on Email and Anti-Spam (CEAS)*, 2004.

[99] S. Moyer and N. Hamiel. Satan is on my friends list: Attacking social networks. `http://www.blackhat.com/html/bh-usa-08/bh-usa-08-archive.html`, 2008.

[100] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia. Exploiting Machine Learning to Subvert Your Spam Filter. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[101] Y. Niu, Y. Wang, H. Chen, M. Ma, and F. Hsu. A quantitative study of forum spamming using context-based analysys. In *Symposium on Network and Distributed System Security (NDSS)*, 2007.

[102] C. Nunnery, G. Sinclair, and B. B. Kang. Tumbling Down the Rabbit Hole: Exploring the Idiosyncrasies of Botmaster Systems in a Multi-Tier Botnet Infrastructure. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2010.

[103] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi. Fluxor: Detecting and monitoring fast-flux service networks. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2008.

[104] A. Pitsillidis, K. Levchenko, C. Kreibich, C. Kanich, G. M. Voelker, V. Paxson, N. Weaver, and S. Savage. Botnet Judo: Fighting Spam with Itself. In *Symposium on Network and Distributed System Security (NDSS)*, 2010.

[105] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*, 1998.

[106] P. Porras, H. Saidi, and V. Yegneswaran. Conficker C analysis. *SRI International*, 2009.

[107] Project Honeypot. `http://www.projecthoneypot.org/`.

[108] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iFRAMEs point to Us. In *USENIX Security Symposium*, 2008.

[109] A. Ramachandran, D. Dagon, and N. Feamster. Can DNS-based blacklists keep up with bots? In *Conference on Email and Anti-Spam (CEAS)*, 2006.

[110] A. Ramachandran and N. Feamster. Understanding the Network-level Behavior of Spammers. *SIGCOMM Computer Communication Review*, 2006.

[111] A. Ramachandran, N. Feamster, and D. Dagon. Revealing Botnet Membership using DNSBL Counter-intelligence. In *USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2006.

[112] A. Ramachandran, N. Feamster, and S. Vempala. Filtering Spam with Behavioral Blacklisting. In *ACM Conference on Computer and Communications Security (CCS)*, 2007.

[113] P. Ratanaworabhan, B. Livshits, and B. Zorn. Nozzle: A defense against heap-spraying code injection attacks. In *USENIX Security Symposium*, 2009.

[114] M. Sahami, S. Dumais, D. Heckermann, and E. Horvitz. A Bayesian approach to filtering junk e-mail. *Learning for Text Categorization*, 1998.

[115] SC Magazine. Accused MegaD operator arrested. `http://www.scmagazineus.com/accused-mega-d-botnet-operator-arrested`, 2011.

[116] D. Sculley and G. M. Wachman. Relaxed Online SVMs for Spam Filtering. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007.

[117] Shadowserver. New fast flux botnet for the holidays. http://www.shadowserver.org/wiki/pmwiki.php/Calendar/20101230, 2011.

[118] S. Shin and G. Gu. Conficker and beyond: a large-scale empirical study. In *Annual Computer Security Applications Conference (ACSAC)*, 2010.

[119] S. Sinha, M. Bailey, and F. Jahanian. Shades of Grey: On the Effectiveness of Reputation-based "Blacklists". In *International Conference on Malicious and Unwanted Software*, 2008.

[120] J. Song, S. Lee, and J. Kim. Spam filtering in twitter using sender-receiver relationship. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2011.

[121] B. Stock, J. Gobel, M. Engelberth, F. Freiling, and T. Holz. Walowdac Analysis of a Peer-to-Peer Botnet. In *European Conference on Computer Network Defense (EC2ND)*, 2009.

[122] B. Stone-Gross, R. Abman, R. A. Kemmerer, C. Kruegel, D. G. Steigerwald, and G. Vigna. The underground economy of fake antivirus software. In *Proceedings on the Economics of Information Security and Privacy (WEIS)*. 2010.

[123] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[124] B. Stone-Gross, M. Cova, C. Kruegel, and G. Vigna. Peering Through the iFrame. In *IEEE Conference on Computer Communications (INFOCOM)*, 2011.

[125] B. Stone-Gross, T. Holz, G. Stringhini, and G. Vigna. The Underground Economy of Spam: A Botmaster's Perspective of Coordinating Large-Scale Spam Campaigns. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2011.

[126] B. Stone-Gross, A. Moser, C. Kruegel, E. Kirda, and K. Almeroth. FIRE: FInding Rogue nEtworks. In *Annual Computer Security Applications Conference (ACSAC)*, 2009.

[127] G. Stringhini, M. Egele, C. Kruegel, and G. Vigna. Poultry Markets: On the Underground Economy of Twitter Followers. In *SIGCOMM Workshop on Online Social Networks*, 2012.

[128] G. Stringhini, O. Hohlfeld, C. Kruegel, and G. Vigna. The harvester, the botmaster, and the spammer: On the relations between the different actors in the spam landscape. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2014.

[129] G. Stringhini, T. Holz, B. Stone-Gross, C. Kruegel, and G. Vigna. Botmagnifier: Locating spambots on the internet. In *USENIX Security Symposium*, 2011.

[130] G. Stringhini, C. Kruegel, and G. Vigna. Detecting Spammers on Social Networks. In *Annual Computer Security Applications Conference (ACSAC)*, 2010.

[131] G. Stringhini, C. Kruegel, and G. Vigna. Shady Paths: Leveraging Surfing Crowds to Detect Malicious Web Pages. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[132] 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 *ACM SIGCOMM Internet Measurement Conference (IMC)*, 2013.

[133] Symantec Corp. State of spam & phishing report. `http://www.symantec.com/business/theme.jsp?themeid=state_of_spam`, 2010.

[134] Symantec. Corp. Rustock hiatus ends with huge surge of pharma spam. `http://www.symantec.com/connect/blogs/rustock-hiatus-ends-huge-surge-pharma-spam`, January 2011.

[135] Symantec Corp. Internet security threat report. `http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf`, 2014.

[136] B. Taylor. Sender reputation in a large webmail service. In *Conference on Email and Anti-Spam (CEAS)*, 2006.

[137] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and Evaluation of a Real-Time URL Spam Filtering Service. In *IEEE Symposium on Security and Privacy*, 2011.

[138] 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 *USENIX Security Symposium*, 2013.

[139] A. Thomason. Blog Spam: A Review. In *Conference on Email and Anti-Spam (CEAS)*, 2007.

[140] Twitter. The twitter rules. `http://support.twitter.com/entries/18311-the-twitter-rules`, 2010.

[141] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011.

[142] S. Venkataraman, S. Sen, O. Spatscheck, P. Haffner, and D. Song. Exploiting Network Structure for Proactive Spam Mitigation. In *USENIX Security Symposium*, 2007.

[143] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. In *Symposium on Network and Distributed System Security (NDSS)*, 2007.

[144] 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 *USENIX Security Symposium*, 2013.

[145] 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 *World Wide Web Conference (WWW)*, 2012.

[146] C. Wilson, B. Boe, A. Sala, K. Puttaswamy, and B. Zhao. User Interactions in Social Networks and Their Implications. In *ACM European conference on Computer systems (EuroSys)*, 2009.

[147] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically Generating Models for Botnet Detection. In *European Symposium on Research in Computer Security (ESORICS)*, 2009.

[148] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming Botnets: Signatures and Characteristics. *SIGCOMM Computer Communications Review*, 2008.

[149] W. Xu, F. Zhang, and S. Zhu. Toward worm detection in online social networks. In *Annual Computer Security Applications Conference (ACSAC)*, 2010.

[150] C. Yang, R. Harkreader, and G. Gu. Die Free or Live Hard? Empirical Evaluation and New Design for Fighting Evolving Twitter Spammers. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2011.

[151] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai. Uncovering social network sybils in the wild. In *ACM SIGCOMM Internet Measurement Conference (IMC)*, 2011.

[152] T.-F. Yen and M. K. Reiter. Traffic Aggregation for Malware Detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2008.

[153] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In *ACM Conference on Computer and Communications Security (CCS)*, 2007.

[154] H. Yu, M. Kaminsky, P. Gibbons, and A. Flaxman. Sybilguard: defending against sybil attacks via social networks. *ACM SIGCOMM Computer Communication Review*, 2006.

[155] A. Zand, G. Vigna, X. Yan, and C. Kruegel. Extracting Probable Command and Control Signatures for Detecting Botnets. In *ACM Symposium on Applied Computing (SAC)*, 2014.

[156] C. C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *ACM Conference on Computer and Communications Security (CCS)*, 2002.