DETECTING MALICIOUS USAGE OF ONLINE SOCIAL NETWORK APPLICATION PROGRAMMING INTERFACES FROM NETWORK FLOWS

by

DAN LI

A THESIS

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Master of Science

December 2019

THESIS APPROVAL PAGE

Student: Dan Li

Title: Detecting Malicious Usage of Online Social Network Application Programming Interfaces from Network Flows

This thesis has been accepted and approved in partial fulfillment of the requirements for the Master of Science degree in the Department of Computer and Information Science by:

Jun Li	Chairperson
Reza Rejaie	Member
Lei Jiao	Member

and

Kate Mondloch	Interim Vice Provost and Dean of the Graduate School

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded December 2019

THESIS ABSTRACT

Dan Li

Master of Science

Department of Computer and Information Science

December 2019

Title: Detecting Malicious Usage of Online Social Network Application Programming Interfaces from Network Flows

While online social networks (OSNs) provide application programming interfaces (APIs) to enable the development of OSN applications, some of these applications, unfortunately, can be malicious. They can be running on the devices for OSN users throughout the Internet, causing security, privacy, and liability concerns to the network service providers of these OSN users.

In this thesis, we study how a network service provider may inspect its network traffic to detect network flows from malicious API-based OSN applications. In particular, we devise a deep learning based methodology to detect network flows generated by malicious API-based OSN applications. We implement this methodology on a testbed, and show that our solution is effective and can accurately label 97.6% network flows from the malicious OSN applications, with only 1.6% false positives.

CURRICULUM VITAE

NAME OF AUTHOR: Dan Li

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene
Beijing University of Posts and Telecommunications, Beijing, China
Hebei University of Technology, Tianjin, China

DEGREES AWARDED:

Master of Science, Computer and Information Science, 2019, University of Oregon
Master of Engineering, Computer Technology, 2016, Beijing University of Posts and Telecommunications, China
Bachelor of Engineering, Software Engineering, 2013, Hebei University of Technology, China

AREAS OF SPECIAL INTEREST:

Network and Security
Deep Learning
Data Science

PROFESSIONAL EXPERIENCE:

UO Graduate Teaching/Research Assistant, Sep 15, 2016 – Jun 15, 2019

Software Development Engineer Intern, Amazon, Jun 17, 2019 – Sep 7, 2019

ACKNOWLEDGMENTS

I would like to especially thank my advisor Prof. Jun Li, who is familiar with my research field and helped me identify research problems, figure out the research methodology in this work, and evaluate this thesis. I would also like to thank my committee members, Prof. Reza Rejaie and Prof. Lei Jiao, for their helpful feedback and suggestions during the preparation of this thesis. Moreover, I thank Prof. Hank Childs and Prof. Boyana Norris who supported me to find my career goal.

Last but not least, I want to thank all my friends, family, faculty and staff at the University of Oregon who supported me in my past three years in Oregon.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 01

II. BACKGROUND AND RELATED WORK 05

III. OVERVIEW 08

IV. PROBLEM SETTINGS AND DEFINITION 12

A. Network Flow 12

B. Project Framework 13

C. Malicious API-based Applications, Benign API-based Applications, Human OSN Behaviors 15

1) Malicious API-based Application Behaviors 15

2) Benign API-based Application Behaviors 17

3) Human OSN Behaviors 18

V. DATA PREPARATION 20

A. Data Preparation Platform 21

B. Synthesis of Malicious API-based Application Behaviors, Benign API-based Application Behaviors and Human OSN behaviors 22

1) Synthesis of Human OSN Behaviors 22

2) Synthesis of Benign API-based Applications Behaviors 24

3) Synthesis of Malicious API-based Applications Behaviors 26

C. Synthesis of Flows for Malicious API-based Applications, Benign API-based Applications and Human OSN Behaviors 29

1) Synthesis of Flows for Malicious and Benign API-based Applications 31

Chapter Page

2) Synthesis of Flows for Human OSN Behaviors 33

3) How We Control Flow Attributes When Generating Synthetic Flows 36

VI. DEEP LEARNING BASED APPROACH OF DETECTING FLOWS FROM MALICIOUS API-BASED APPLICATIONS 38

A. Model Selection 38

B. Ground Truth Labeling... 38

C. Dataset Size 39

D. Flow Data Preprocessing 40

E. CNN Model Structure... 42

VII. EVALUATION 45

A. Test Set Size and Evaluation Metrics 45

B. Overall Detection Performance for All Test Set 46

C. Detection Performance for Malicious API-based Applications with Different Post/Comment Frequencies 47

D. Detection Performance for Each Malicious API-based Applications with Different Post/Comment Frequencies.. 49

E. Detection Performance for Detecting Real World API-based benign Applications and Malicious Applications.. 52

VIII. DISCUSSION 55

A. Is A Single Timing Pattern Feature Enough to Detect Flows Generated by Malicious OSN Applications? 55

Chapter Page

B. Can This Project Only Detect Flows from Malicious OSN Programs That Demonstrate a Similar Timing Pattern with Our Five Used Malicious Timing Patterns? 61

IX. CONCLUSION 64

X. FUTURE WORK 65

REFERENCES CITED 67

LIST OF FIGURES

Figure Page

1. Network flow traffic 13
2. Setting of a social network service provider (NSP) in detecting malicious OSN network flows.. 14
3. Malicious OSN application controlled accounts' behavioral time points show five malicious patterns 16
4. Benign API-based application controlled OSN accounts' behavioral time points follow a benign pattern 18
5. Streamline of how different events switch to each other on WordPress 24
6. Simulation of 4 benign API-based applications by changing Poisson parameters 25
7. Four simulation results for the first malicious API-based applications................ 29
8. Converted images for human, malicious application and benign application flows 42
9. CNN model structure 43
10. Detecting performance for predicting all test sets 47
11. Detecting performance for malicious API usage with different posting/commenting times per day 48
12. Comparison for malicious API application #1posting/commenting 10 and 50 times, respectively.. 49
13. Detection result for each malicious API-based application posting/commenting 10 to 50 times per day 50

Figure Page

14. Comparison for bad1 application and bad2 application for posting/commenting 10 times per day 51

15. Detecting results for flows generated by three real world malicious API-based OSN applications 53

16. Detecting results for flows generated by three real world benign API-based OSN applications 53

17. Detecting results for predicting the test set for the model with only the timing feature application 56

LIST OF TABLES

Table Page

1. Human operations on OSN 23

2. Aggregated flows number for each OSN behaviors 39

3. Aggregated flows number for applications with different malicious/benign application patterns 40

4. Test set data size for each behavior 45

5. Detection accuracy result for different flows in each timing based model........... 57

6. Detection accuracy result for different flows in different feature based model 60

7. Detection accuracy result for different flows in each four-feature based model... 61

CHAPTER I

INTRODUCTION

Online social networks (OSN) have become extremely popular with an ever-growing user base. At the time of writing this master thesis, Facebook, Twitter, and WeChat each has 2.2 billion, 0.4 billion, and 0.5 billion users, respectively. In particular, in order to further enrich and improve the user experience, OSNs have provided public Application Programming Interfaces (APIs) to enable the development of OSN applications that can access OSN data and functions. However, the provision of these APIs can cause severe security concerns.

Whereas these public APIs make it easy and convenient for OSN applications to provide various legitimate OSN services, such as querying an OSN user's profile information and friend lists, retweeting certain tweets, or making automated comments, they may also be abused or misused by malicious OSN applications. They can be running on the devices for OSN users throughout the Internet, causing security, privacy, and liability concerns to the network service providers of these OSN users. Very often, by using OSN APIs, a malicious OSN application may control bot accounts to post or reply with spam or fraudulent information, run a crawler to collect private and sensitive OSN user data, or act as a third-party application to obtain access to accounts of OSN users, followed by collecting the profiles of these users and even their friends. In a widely known case, Facebook was reported to leak data of up to 87 million users through a third-part psychology quiz application (NC Matthew, 2018).

While abusing OSN APIs, these malicious OSN applications particularly cause concerns to network service providers (NSPs) for OSN users. It could be an Internet service provider (ISP), an enterprise or campus network. If a malicious OSN application is running inside a network, it can imply that one or multiple machines in the network are compromised, the application can subvert the privacy of OSN users, and the network may have to be liable for the security and privacy violations.

However, an NSP is not at the same position as an OSN provider to deal with malicious OSN applications. An OSN provider can try to monitor API calls, obtain full knowledge of user profiles and posts, as well as access the entire OSN graph, in order to detect OSN spam accounts (X Zheng, 2015), (F Benevenuto, 2010), (A Almaatouq, 2016), (M Fazi, 2018), limit large-scale crawling activities (M Mondal. 2012), detect malicious third-party OSN applications (SH Ahmadinejad, 2013), and so on. On the other hand, an NSP only has limited knowledge of OSN data (such as user posts, profiles, social behaviors, OSN graphs). It can only access the traffic across its network, thus not able to leverage the aforementioned existing work toward detecting malicious OSN applications.

We therefore study how an NSP may monitor its traffic to detect traffic flows from malicious OSN applications. We make the following contributions:

1) We define a problem of detecting flows from malicious API-based OSN applications, whereas the flow data used do not include traffic payload. In another word, the problem assumes no knowledge of OSN topologies or specific user profiles and data.

2) We propose a solution to detect flows from malicious API-based OSN applications. First, we train a deep learning model for malicious OSN flow detection based on three types of OSN flows: flows from malicious API-based OSN applications, flows from benign API- based OSN applications, and flows from human user operations on the OSN. For each machine running inside an NSP, we extract, aggregate, normalize and visualize flows generated between the machine and an OSN, then apply our trained model to determine whether the normalized flows are generated by a malicious OSN application running on a machine inside the NSP.

3) We implement our proposed solution on a testbed, where we simulate and collect flows for various malicious OSN applications, benign OSN applications, and human user operations. The trained deep learning model is able to detect flows generated by malicious OSN applications with high accuracy and low false positive. In particular, the trained model is able to label flows from three real-world benign OSN applications and three real-world malicious OSN applications with high accuracy. Our research demonstrates that it is feasible to detect flows from malicious API-based applications on OSNs. What's more, our proposed solution can apply to any other social networks. e.g. Facebook, Twitter.

The rest of this thesis is organized as follows. After Chapter II about related work, we first overview the problem and our solution in Chapter III, followed by a formal description of the problem in Chapter IV. We then generate flows on a test bed in Chapter V to implement our solution, describe our detailed solution in Chapter VI, evaluate the performance of our solution in Chapter VII, discuss our work in Chapter VII, conclude our work in Chapter IX, and introduce our future work in Chapter X.

CHAPTER II

BACKGROUND AND RELATED WORK

While APIs released by OSNs are supposed to work for third-party developers to access OSN services, researchers have shown they can be easily misused by crawlers to crawl OSN data or by spammers to spread fraud or spam content ((AH Wang, 2010), (A Saroop, 2011) and (CM Zhang, 2011)). The work in (AH Wang, 2010) crawled a large amount of sensitive OSN data by using OSN APIs, and research in (A Saroop, 2011) even designed an API-based crawler that attackers can use to crawl a large amount of Twitter network structural information. At the same time, spam accounts controlled by malicious API applications are common on OSNs; as the research in (CM Zhang, 2011) points out that, many automated spam accounts on OSNs prefer to use API rather than a web browser to spread fraud or spam content.

There are certain proposed methods that detect malicious automated spam accounts on OSNs, including recent work in (X Zheng, 2015), (F Benevenuto, 2010), (A Almaatouq, 2016), and (M Fazi, 2018). Basically, they all analyze the post content, user profiles, or social behaviors of spam accounts and rely on these features to detect spam accounts. However, a network service provider that usually only collect network flow data can hardly have access to such features, thus not able to employ such a method to detect OSN spam accounts inside their network.

Methods are also proposed to prevent crawling activities on OSNs. Research in (J Herrera-Joancomartí, 2011) and (M Mondal, 2012) proposed countermeasures to prevent attackers from crawling sensitive OSN user data. Research in (J Herrera-Joancomartí,

2011) proposes an “Online Social Honey net” concept by deploying a set of users on network to attract and defend OSN crawler attackers, but it only proves the feasibility of using this concept to prevent crawlers, not about deploying it in the real world. Research in (M Mondal, 2012) proposes a Genie system which is deployed at OSN providers to thwart crawlers by detecting their different browsing patterns. This work analyzes user traces of visiting their friends and non-friends, which is sensitive and, again, not accessible in network flow data, so their methodology is not usable by a network service provider to detect malicious OSN activities.

On the other hand, many methods have been proposed to analyze flow data to detect network attacks or anomalies. Research in (R Singh, 2012) uses campus traffic flows to detect anomaly broadcast traffic, while (J. François, 2011) extends the popular PageRank algorithm to detect botnet traffic. The work (A King, 2009) and (P Barfor, 2001) both detect network traffic flow anomalies by analyzing flow-level anomaly features. And research in (ZQ Wang, 2008), (H Rick, 2013), and (VDS Daniël, 2015) also proposes several real-time intrusion detection systems based on monitoring network flow traffic. These papers all follow a similar idea by detecting a specific attack based on the flow- level features of the attack. However, the specific attack features explored in these existing methods are different from the features of OSN attacks, making their feature-based detection methods basically ineffective in detecting OSN attacks.

In obtaining a better understanding how users use or interact with online social networks, research in (B Fabrício, 2012), (W Watcharee, 2012), (M Zoltán, 2011), (S Fabian, 2009) investigated how to use traffic analysis methods to study social networks. However, these methods analyze network traffic that includes packet payload in general, which is different from our work that uses the network flow data that only carry aggregated packet header information.

CHAPTER III

OVERVIEW

The goal of this project is to detect flows from malicious API-based applications on OSNs. We assume that traffic flows of human OSN behaviors and benign API-based application behaviors share some normal flow-level features, while traffic flows of malicious API-based applications have some malicious flow-level features. We propose a solution to detect flows generated by malicious API-based applications. Our solution first aggregates flow for the above three category flows separately, normalizes each of the aggregated flows, then uses normalized flows as the input to train a deep learning model. The trained deep learning model can learn features effectively from high-dimensional network flows, and can label flows from malicious API-based applications. We implemented this solution on a test bed.

Since there is no available dataset providing flow data generated by malicious API-based applications and other benign OSN flows, we use emulation to generate those flows. We deploy a small social network, WordPress, as a test bed where users can browse and reply to each other's posts. We then collect flows for human OSN behaviors, benign API- based applications, and malicious API-based applications on this small social network. We performed these three behaviors on a client which is installed with flow generation software and flow collection software. In this way, when we simulate three behaviors on this client, their corresponding flows will be collected on the client side by flow collection software.

To simulate a large amount of various human OSN behaviors, we write scripts to emulate various possible human behaviors on a social network website. These operations include login, post, comment, browsing behaviors, and so on. Human behaviors on social networks are driven by a series of click events. We use scripts to emulate those click events instead, and flows generated by script-controlled click events should be the same as flows generated by human-controlled click events. For benign API-based applications, OSN APIs should only be a tool used by human beings. Such tools will post content on social networks for human beings whenever people want to. Therefore, those benign API-based applications' behavior time points should follow human post/comment/like timing patterns on OSN. Human behavior patterns can be modeled as different Poisson processes (CM Zhang, 2011), so benign API-based application behavior time points can also be modeled as different Poisson processes. We simulate various benign API- based application behavior time points as different Poisson processes by changing Poisson distribution parameters. At each behavior time point, the benign OSN application will post some useful content. For malicious API-based applications, we write five known malicious application scripts, and their behavior time points are decided by these five known malicious API-based applications' behavioral patterns (CM Zhang, 2011). Each application posts spam or malicious posts, comments following one of five known malicious timing patterns. We model each malicious API-based application behavioral timing pattern as a combination of different probability distributions, and simulate each malicious pattern by changing all related parameters. To collect flow data generated by benign or malicious API-based applications and human OSN behaviors, we collect corresponding flows when we simulate those three behaviors on the client side.

To save time generating data for human OSN behaviors, benign and malicious API-based applications, we synthesize flows for each of those three behaviors. When people or applications are active and perform various operations on OSNs, there are some flows generated. Otherwise, no flows are generated. To synthesize flows generated in a long period of time, we only need to combine flows generated when people or applications are active on OSN together. First, we collect flows when applications or people are active on OSNs, then combine those flows together with proper intervals. Interval length indicates how long people or applications wait between two active sessions. In this way, we can synthesize flows generated in a long period of time for benign and malicious API-based applications, and human OSN behaviors.

We've generated flows for the above three behaviors. After that, we extracted all flows generated by the communication between the client and our small social network by extracting flows whose source IPs or destination IPs belong to our deployed OSN server. We've found that even a very single user operation behavior, such as a click behavior, can generate several flows, so a single flow data carries very little information about how users behave on an OSN. Therefore, we aggregate flows that occurred in a pre-defined time window together, and those aggregated flows can carry more information about how a user behaves in the pre-defined time window. For each aggregated flow group, we transformed it into an image which carries the main information of the aggregated flow group. Those transformed images act as the input to train the deep learning model, which can automatically find potential flow-level features for human OSN behaviors, benign and malicious API-based applications. After the model is well-trained, it is able to detect these flows generated by malicious API-based applications with high accuracy, precision and recall scores.

The methodology described above enables NSPs to obtain a deep learning model with very good performance in detecting flows generated by malicious API-based applications. We built a small social network through the WordPress blog system as a testbed, and implemented our proposed solution to detect flows from malicious API-based applications on this testbed. Furthermore, our proposed methodology not only works on the deployed testbed, but can also be applied to other OSNs (Twitter, Facebook, Instagram) to detect flows from any malicious API-based applications for any NSPs.

CHAPTER IV

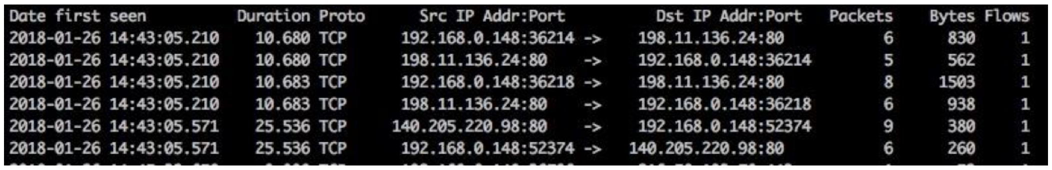
PROBLEM SETTINGS AND DEFINITION

Now that we have introduced the background and motivation for this project, and given an overview of our work in Chapter III, we can now provide a more specific problem formulation. In this chapter, we begin by introducing network flow data. After that, we explain how we define malicious API- based applications, benign API-based applications and human OSN behaviors. Our purpose in this project is to detect malicious API-based application behaviors based on network flows. Finally, we summarize the problem to be solved in this project.

A. Network Flow

Current network communication is based on packet switching. Traffic flow is a sequence of aggregated packet headers from a source computer to a destination computer. Flows are identified by a five-tuple key, including the IP protocol, source computer's IP address and port number, and destination computer's IP address and port number. There are several attributes used to describe a flow, and those attributes are extracted from packet headers, such as the flow start timestamp, end timestamp, source IP and port number, destination IP and port number, the IP protocol and the protocol's attributes. If the TCP protocol is used, the flow may also include the TCP flag as its attribute. Flow attributes can also include statistical numbers for total bytes and total packets received by each flow. A flow data can have any attribute which can be read from packet headers. In this thesis, we use the widely used NetFlow (B Claise, 2004) format of network flow traffic. Figure 1 shows several examples of the NetFlow data.

Based on our description, the flow data can carry very limited sensitive social network user data, and it could be challenging to detect malicious social network flows for API-based applications. However, our research proves that it's feasible to detect malicious API-based applications generated flows.



```
Date first seen          Duration Proto      Src IP Addr:Port          Dst IP Addr:Port   Packets    Bytes Flows
2018-01-26 14:43:05.210    10.680 TCP      192.168.0.148:36214 ->    198.11.136.24:80            6      830     1
2018-01-26 14:43:05.210    10.680 TCP      198.11.136.24:80    ->    192.168.0.148:36214         5      562     1
2018-01-26 14:43:05.210    10.683 TCP      192.168.0.148:36218 ->    198.11.136.24:80            8     1503     1
2018-01-26 14:43:05.210    10.683 TCP      198.11.136.24:80    ->    192.168.0.148:36218         6      938     1
2018-01-26 14:43:05.571    25.536 TCP     140.205.220.98:80    ->    192.168.0.148:52374         9      380     1
2018-01-26 14:43:05.571    25.536 TCP      192.168.0.148:52374 ->   140.205.220.98:80            6      260     1
```

Figure 1. Network flow traffic

B. Project Framework

In this part, we will introduce the framework of this project and answer some basic questions, such as how malicious or benign OSN applications and humans within a NSP communicate with OSN servers, where OSN network flow data inside a NSP can be caught, and where our proposed malicious flows detection method should be deployed to detect malicious flows for a NSP.

Figure 2 shows the framework of this project. Our purpose is to detect flows generated by malicious OSN applications for NSPs. NSPs provide network services to their users, and a NSP could be part of an Internet Services Provider (ISP). For example, a company network is a NSP, because it can provide network services to all workers in its company.

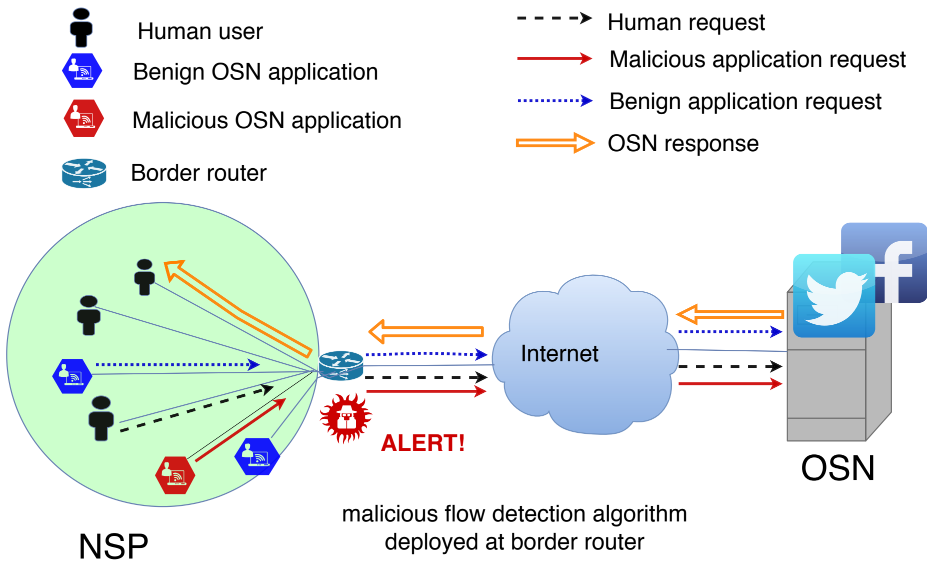

Human user
Benign OSN application
Malicious OSN application
Border router
Human request
Malicious application request
Benign application request
OSN response
Internet
ALERT!
OSN
NSP
malicious flow detection algorithm deployed at border router

Figure 2. Setting of a network service provider (NSP) in detecting malicious OSN network flows

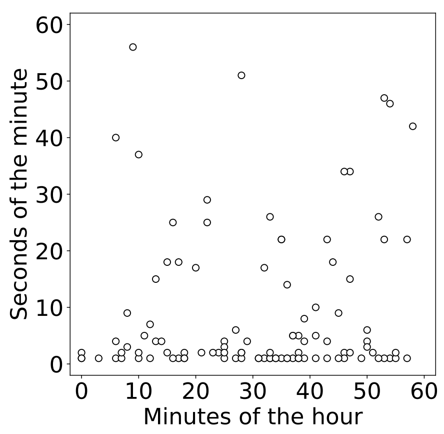
Human, benign OSN applications and malicious OSN applications in some NSPs can send requests to OSN severs. When OSN servers receive their requests, OSN servers will reply responses to the people or applications who sent requests in previous NSPs. Both requests from the NSP to OSN servers and responses from OSN servers to the NSP will go through the NSP's border router. Therefore, the border router can collect flows generated between the application or human within its network and the OSN servers. Our malicious flow detection method can be deployed at the border router of the NSP. If there are some malicious flows detected at the border router, it can send some alerts to the NSP.

C. Malicious API-based Applications, Benign API-based Applications and Human OSN Behaviors

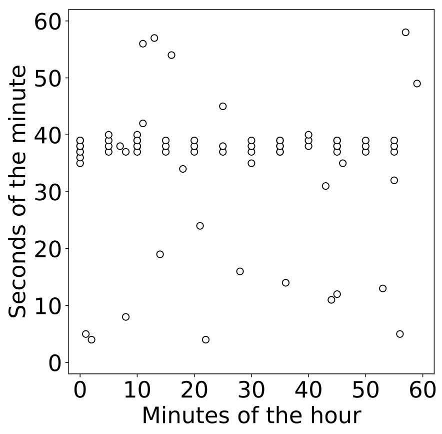
1) Malicious API-based Application Behaviors

Attackers usually run malicious API-based applications to frequently collect private OSN data or spread spam information on OSNs. Whether crawling behavior is benign or malicious can only be decided by how people use or analyze crawled data, and this is not our focus. In this project, we do not classify them as malicious only based on their crawling behaviors. This project mainly focus on detecting malicious API-based spammer applications on OSNs. The malicious behaviors to be detected are when malicious OSN API-based applications post or comment with spam content (post with malicious URLs for malwares, or spam advertisements) on OSNs. E.g. "Free Business Links For Chemical Suppliers at http://catalogs.indiamart.com/category/chemicals-fertilizers.html".

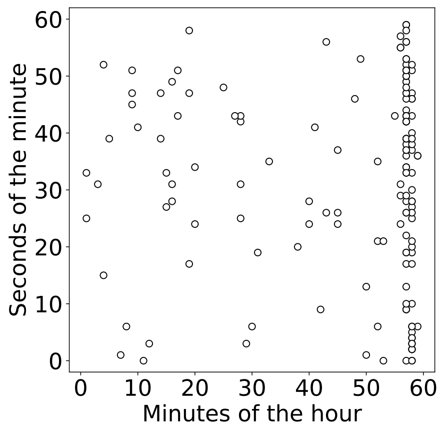
Malicious API-based spam applications behave with malicious purposes, so their controlled benign accounts' behavioral patterns are probably different from benign accounts. There are five known malicious patterns for malicious API application-controlled spam/bot accounts on Twitter (CM Zhang, 2011). Those accounts' post/retweet/like behavior time points on Twitter follow five malicious patterns in Figure 3. Therefore, corresponding malicious API-based applications' post/retweet/like time points should also follow those five known malicious patterns. For those five malicious patterns in Figure 3, the x axis indicates minutes of the hour for an account's post/comment/like/retweet behavior time point, while the y axis indicates the seconds of the minute for this account's post/comment/like/retweet behavior time point.



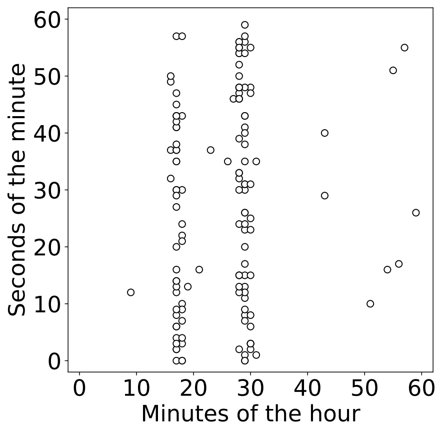
(a) Fixed malicious second pattern



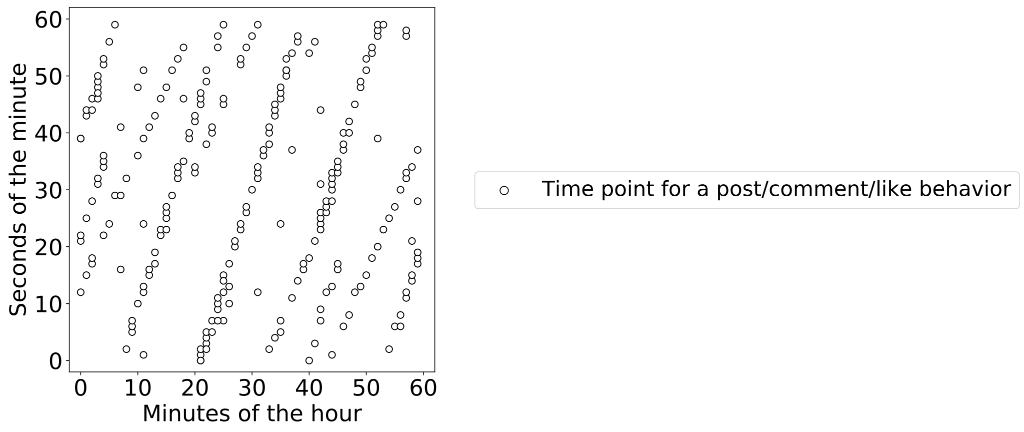
(b) Fixed malicious interval pattern



(b) Fixed malicious minute pattern



(d) Two fixed malicious minutes pattern



(e) Hybrid malicious pattern

Figure 3. Malicious OSN application controlled accounts' behavioral time points show five malicious patterns

Attackers usually use OSN APIs to spread spam information by posting and commenting. In this project, we mainly focus on detecting malicious OSN applications injecting spam contents into the network. In this situation, we define the two most common malicious API-based behaviors:

(1) API-based post or comment with spam content for a single account.

(2) API-based change accounts to post or comment with spam content.

In this project, if any of those three API-based behaviors post spam posts or comments following any of five known malicious API usage patterns, we define it as a malicious API-based application behavior.

2) Benign API-based Application Behaviors

Some benign OSN API-based applications may use OSN-released APIs to provide people useful information, such as real-time weather warnings, earthquake information for a specific location, and news happening all around the world. In this project, we define benign OSN application behaviors are when OSN API-based applications post useful information to people.

When OSN-released APIs are used for a benign purpose, public APIs are supposed to provide OSN services on those benign applications and work automatically for human beings with benign purposes whenever people need to post/comment/like. For example, "Flow" is a Microsoft application which integrates the Facebook API, Twitter API and Instagram API. It can help people post on Facebook, Twitter, Instagram at the same time when people want to publish their stories on multiple social networks. Since APIs are only tools for human beings to post and comment on OSNs, benign API usage behaviors should follow human beings' post and comment patterns. It has been found that human post/comment/like time points on Twitter can be modeled as different Poisson Processes (CM Zhang, 2011). This benign pattern can be converted to the pattern in

Figure 4 which is drawn in a similar way to Figure 3's depiction of five malicious patterns based on humans' post/comment/like/retweet time points. The benign API-based application's behavioral time point patterns should also follow this benign human post/comment pattern on OSNs.

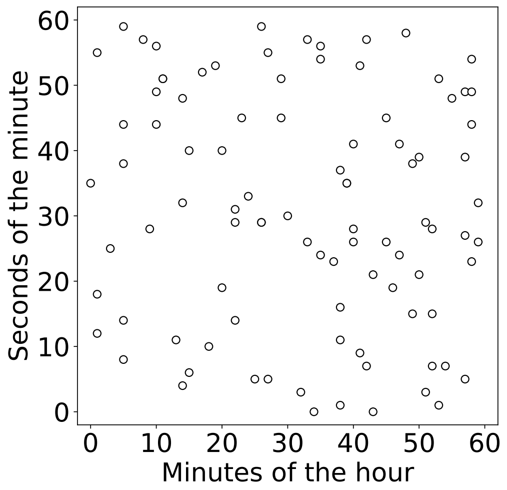


Figure 4. Benign API-based application controlled OSN accounts' behavioral time points follow a benign pattern

Therefore, in this project, if an API-based application posts benign posts/comments following this benign Poisson pattern, we define it as a benign API-based application.

3) Human OSN Behaviors

In most cases, users visit social networks and perform some normal operations using browsers instead of APIs. Human beings usually access OSNs on browsers to do various operations, such as login, post, comment, browse and so on. Most OSN flows are generated by human operations on OSNs. In this project, we define various human-based browser operations on OSNs as normal behaviors, and define flow data generated by human operations as benign flows.

To summarize, we define three behaviors in Chapter IV: malicious API-based application behaviors, benign API-based application behaviors and human OSN behaviors. If APIs are used by applications to spread malicious information, then their behaviors follow five known bad timing patterns. If APIs are used by applications to spread benign information, then their behaviors follow the benign timing pattern. Humans can perform various operations on OSNs, and we define all human behaviors as benign. Our purpose is to detect flows for malicious API-based application behaviors from flows generated by all three behaviors. We assume that traffic flows of human OSN behaviors and benign API-based application behaviors share some benign flow-level features, while traffic flows of malicious API-based applications have some malicious flow-level features. We aim to detect flows of malicious API-based applications by detecting flows with malicious features.

CHAPTER V

DATA PREPARATION

Since there is no existing and available data set providing flows for malicious API-based application behaviors or benign OSN behaviors, we need to generate those data by ourselves. As described in the problem formulation from Chapter IV, we define three OSN behaviors in this project: malicious API-based application behaviors, benign API-based application behaviors, and human OSN behaviors. We will generate flows for those three behaviors.

To generate flows for malicious API-based application behaviors, we have each malicious API-based OSN behavior simulate one of five malicious API usage timing patterns in posting malicious contents or malicious commenting contents by setting related parameters. To generate flows for benign API-based application behaviors, we have different API-based OSN behaviors simulate benign API usage timing patterns to post benign information by setting related parameters. In addition, we simulate all kinds of human operations on OSNs using browsers, and collect those generated flows as benign human OSN behavior flows.

In this chapter, we will start by introducing the data preparation platform, then simulate malicious API-based application behaviors, benign API-based application behaviors and human OSN behaviors. After that, we describe how we synthesize flows for three behaviors respectively.

A. Data Preparation Platform

As described in research method chapter, we deploy a small social network WordPress in order to replay our synthesized malicious API-based application behaviors, benign API-based behaviors and human OSN behaviors on this small social network.

The small social network is a blog WordPress, where people can post and reply to each other. It is deployed on a reserved server on DigitalOcean with a fixed IP address 165.227.20.24. The server is configured with 512 MB Memory, 20 GB Disk, and Ubuntu 16.04.3 x64 Operating System. The client is a Dell laptop configured with 8GB Memory, 128GB Disk, and Ubuntu 16.04 Operating System.

We simulate malicious API-based application behaviors, benign API-based behaviors and human OSN behaviors on the client. The client is installed with traffic flow generation software softflowd and flow collection software nfdump. When we are simulating malicious API-based application behaviors, benign API-based behaviors and human OSN behaviors on the client separately, the flow data generation software softflowd and collection software nfdump are running on this client and collecting corresponding flow data for each behavior. In flow data generation and collection process, we collect flows mainly based on the below three defaulted flow collection parameters.

(1) Maxlife value is set to 604800s. The value of this parameter is the maximum lifetime that a flow may exist for. The client and server may keep communicating to each other for a time longer than Maxlife. When the connection duration reaches to Maxlife, the current flow for the current connection will expire, and a new flow will be generated for describing this connection.

(2) Expint value is set to 60s. This parameter of Expin specifies the interval between expiry checks. This is to say, when the client and server are sending packets to each other, if the interval of two consecutive packets is within Expint time, they will be classified into the same flow. If the interval of two consecutive packets is larger than Expint time, they will be classified into different flows.

(3) Flows have two directions: from source to destination, and from destination to source. If the traffic exceeds 2 Gib in either direction, then the corresponding flows will expire, and a new flow will be generated for continuous connections.

Based on the above built testbed and its set up environment and parameters, we collect flows for malicious OSN API-based applications, benign OSN API-based applications and human behaviors.

B. Synthesis of Malicious API-based Application Behaviors, Benign API-based Behaviors and Human OSN behaviors

1) Synthesis of Human OSN Behaviors

Since most OSN flows are generated by normal human operations on OSNs, it is very important to synthesize normal human beings' behaviors accurately. We can then collect human OSN behavior flows accurately as ground truth.

In order to simulate normal human beings' behaviors comprehensively, we write scripts to simulate all kinds of human user behaviors on WordPress, e.g. login, browse, post, comment by browser. Table 1 shows all human basis operations on WordPress. Human beings' operations on OSNs are driven by a series of click events on the browser. To simulate human behaviors accurately, the click event stream pattern in our scripts of human operations on WordPress follows the click pattern summarized in a real Chinese social network RenRen (G Wang, 2017).

Table 1. Human operations on OSN

Category	Event type
Account	Login
Browse	View a post (go to the post webpage, then go back to previous page or main page)
	Browser feeds (scroll mouse, may go next page, or view a post, then go back to previous page or main page)
	Return to previous page
	Go to main page
	Go to next page
	View a recent post
	View a recent comment
Comment	Login in then comment, no login and comment as a stranger
Post	Login then post

Human behaviors on OSN can be summarized as a series of events switching to each other. Figure 2 visualizes the logic of how different events switch to each other. We write scripts to simulate 27 streamlines of human behavior chains based on event occurrence sequences, which cover almost all possible user operation streamlines on WordPress in the real world except infinite looping browsing posts.

For each streamline, we simulate various possible streamline implementations in the real world by changing parameters to simulate its operation logic. For example, one of the streamline is: login, then view page 0-10 separately. To simulate this streamline, after a user finishes the login step, he or she may browser page 0, pages 0 to 1, pages 0,1, 2,...., page 0,1,2 ...10 separately. In this way, we simulate nearly all possible implementations for this streamline by changing page parameters. For all remaining streamlines, we simulate various implementations by changing related parameters, so our simulation covers nearly all possible human behaviors on this small social network accurately and comprehensively.

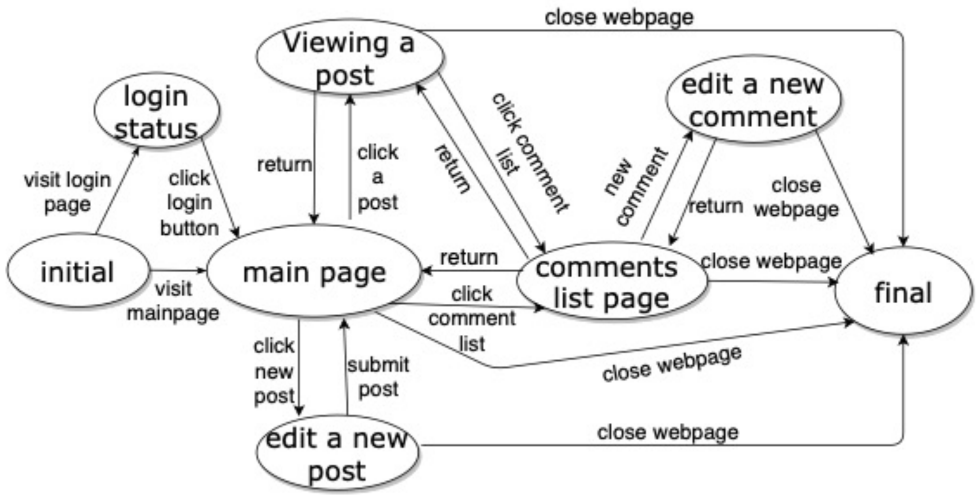
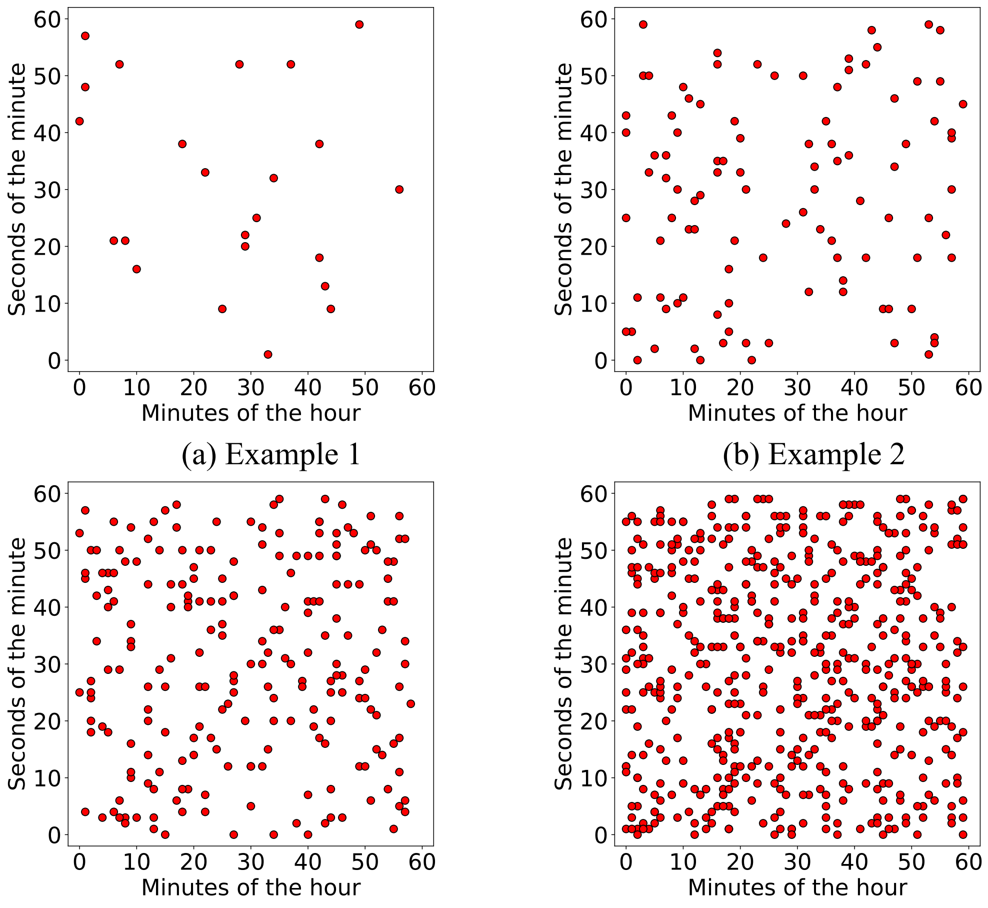


Figure 5. Streamline of how different events switch to each other on WordPress

2) Synthesis of Benign API-based Application Behaviors

We will introduce how we synthesize benign OSN API-based application behaviors in this part. APIs provided by OSNs are supposed to be used for third party developers so they can integrate OSN services to their own developed software. Third party software are supposed to use API-provided OSN services to serve human beings automatically when people need their functions. Therefore, the benign API third party applications' behavioral timing patterns should follow human post/comment/like patterns on OSN. It's known that the timing pattern of how human post/comment/like on Twitter can be modeled as different Poisson Processes (CM Zhang, 2011), so benign API-based third party applications' behavioral time points should follow these Poisson Processes. Besides API-based benign third party applications, there are some benign API-based bot applications on OSNs, and those bots could provide helpful weather warnings or broadcast useful news to the general public. Benign bot applications will post content on a social network whenever there is news or a warning, and those news and warnings occurrence patterns can also be modeled as various Poisson Processes, so benign API-based bot applications' behavioral patterns can also be modeled as Poisson Processes. Thus, all benign API-based applications' behaviors should follow the Poisson Distribution.



(a) Example 1 (b) Example 2

(c) Example 3 (d) Example 4

Figure 6. Simulation of 4 benign API-based applications by changing Poisson parameters

If API-based applications post benign contents following this benign Poisson pattern in Figure 4, we define it as benign API usage behaviors. To simulate all possible benign API-based application behaviors, we crawl benign posts' data from benign automatic accounts on Twitter, and simulate posting benign information behaviors. Their behavior time points follow various benign Poisson processes by changing different parameters. Figure 6 shows four examples of our simulated benign API-based application behavioral timing patterns with four different Poisson parameters.

3) Synthesis of Malicious API-based Application Behaviors

Malicious OSN API-based applications behave with malicious purposes to post malicious data, and those applications' behavior patterns are probably different from benign accounts. As we discussed in Chapter IV, there are five known malicious API-based application behavioral timing patterns. We have also defined two categories of the most common API-based behaviors as API-based post or comment with spam contents for a single account or API-based change accounts to post or comment with spam contents.

We download a Twitter dataset which provides malicious content that were posted or commented by malicious spam accounts. To simulate various malicious API-based application behaviors, we have the behavior time points for each API-based OSN behavior decided by five known malicious timing patterns. The application will post or comment malicious content at each behavior time point. We model each malicious timing pattern as combinations of different probability distributions, and simulate each malicious timing pattern by changing related parameters for corresponding probability distributions.

We take the first malicious API-based malicious pattern as an example. Based on observations, the probability density function of the first malicious timing pattern can be modeled as the combination of a uniform distribution ($P0$) and a group exponential distributions ($\lambda e^{-\lambda t}$) with the same λ value. Therefore, we can obtain the first malicious timing pattern probability density as formula 1. In formula 1, $P0$ is the possibility density for the uniform distribution, while $\lambda e^{-\lambda(t-dT*k-T0)}$ indicates probability densities for multiple exponential distributions. The parameter $M0$ can adjust the weight of all exponential distribution densities. dT is the time step length between each two adjacent exponential distributions, while $T0$ is the initialized time step length for the first exponential distribution. Parameters λ can decide the shape for all exponential distributions.

To simulate various implementations of the first malicious API usage timing patterns, we change parameters $\lambda, P0$, $M0$, dT and $T0$ in its density function, then use the density function with various changed parameters to simulate different first malicious API usage patterns. Figure 7 shows four simulation results for the first malicious API usage timing pattern with different parameters.

In a similar way, the probability density of the second malicious timing pattern can be modeled as a combination of a uniform distribution and a group normal distributions with the same δ. The third malicious API usage timing pattern probability density is a combination of a uniform distribution and a group of Poisson distributions with the same δ. The fourth malicious API usage timing pattern can be modeled as a combination of a uniform distribution and two group Poisson distributions with parameters $\lambda 1$ and $\lambda 2$ separately, while the probability density for the fifth malicious timing pattern can be modeled as a combination of a uniform distribution and two group normal distributions with parameters $\delta 1$ and $\delta 2$ separately. All probability density functions for the five malicious API usage patterns are shown in formula 1, 2, 3, 4 and 5 respectively.

$$f1(t,\lambda) = P0 + M0 \sum_{k=0}^{k=n} \lambda e^{-\lambda(t-dT*k-T0)} \tag{1}$$

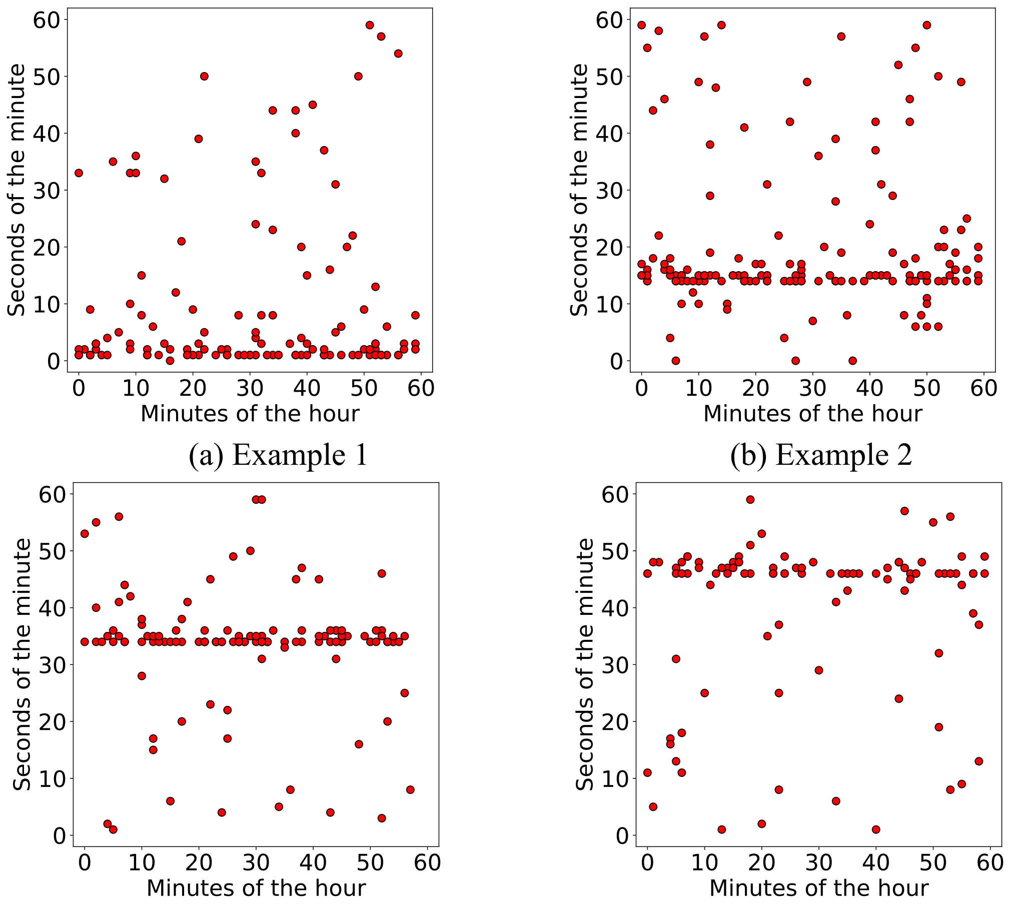
$$f2(t,\delta) = P0 + M0 \sum_{k=0}^{k=n} \frac{1}{\delta\sqrt{2\pi}} e^{-\frac{(t-dT*k-T0)^2}{2\delta^2}} \tag{2}$$

$$f3(t,\lambda) = P0 + M0 \sum_{k=0}^{k=n} \frac{e^{-\lambda}\lambda^{(t-dT*k-T0)}}{(t-dT*k-T0)!} \tag{3}$$

$$f4(t,\lambda1,\lambda2) = P0 + M1 \sum_{k=0}^{k=n} \frac{e^{-\lambda1}\lambda1^{(t-dT1*k-T1)}}{(t-dT1*k-T1)!} + M2 \sum_{k=0}^{k=n} \frac{e^{-\lambda2}\lambda1^{(t-dT2*k-T2)}}{(t-dT2*k-T2)!} \tag{4}$$

$$f5(t,\delta1,\delta2) = P0 + M1 \sum_{k=0}^{k=n} \frac{1}{\delta1\sqrt{2\pi}} e^{-\frac{(t-dT1*k-T1)^2}{2\delta1^2}} + M2 \sum_{k=0}^{k=n} \frac{1}{\delta2\sqrt{2\pi}} e^{-\frac{(t-dT2*k-T2)^2}{2\delta2^2}} \tag{5}$$

As we described above, there are two categories of the most common API-based malicious behaviors, each behavior's post or comment contains malicious content, and their behavioral time points are decided by any of five malicious API usage timing patterns. To simulate each malicious timing pattern accurately, we build a probability function for it, and simulate various implementations for this malicious timing pattern by changing related parameters in its corresponding probability function. We have each API-based behavior's time points follow five malicious patterns separately by changing corresponding probability function parameters for each malicious pattern. In this way, we can simulate various malicious API-based applications' posts or comments of various malicious content with five known malicious patterns, and our simulation result is able to cover nearly all possible five known malicious API-based application behaviors.



(a) Example 1 (b) Example 2

(c) Example 3 (d) Example 4

Figure 7. Four simulation results for the first malicious API-based application

C. Synthesis of Flows for Malicious API-based Applications, Benign API-based Applications and Human OSN Behaviors

In the above Chapter V, we have introduced how we simulate malicious API-based applications, benign API-based applications and human OSN behaviors to generate corresponding flow-level data. In this part, we will describe how we synthesize flows for malicious API-based applications, benign API-based applications and human OSN behaviors. To begin with, we explain why it's necessary to synthesize flows for those three behaviors. After that, we introduce more details about how to synthesize flows for these three behaviors.

To speed up ground truth generation time, we synthesize flows for malicious API-based applications, benign API-based applications and human OSN behaviors. We simulate users' behaviors in both user active time and user waiting time. During user active time, users visit OSNs actively, and perform various operations on OSNs. During user waiting time, users don't access OSNs and don't do any operations on OSNs. To generate ground truth, we write scripts to simulate user behaviors both in their active time and in their waiting time. However, for both benign user accounts and malicious user accounts, the waiting time is much longer than the active time. Therefore, the speed of ground truth generation is very slow because a lot of time is wasted on simulating user waiting time.

However, during user waiting time, there are no corresponding flows generated because users don't communicate with OSNs during waiting time. In addition, based on observations, we find that a single API or human behavior on OSN can generate several flows. Even if API or human behavior time points are very close, an identical API or human behavior occurring at different time points can generate different flows. Not only could flow numbers generated by the same behavior be different, but also the attribute values of each flow are different, such as flow start time, flow number, flow duration, TCP flag, packets number, bytes, and so on. Therefore, flows generated by a single behavior at different time are independent and different. Since there are no flows generated during waiting time, we can synthesize flows for malicious and benign behaviors by combining flows of different OSN behaviors together and only change flows' generation time attributes.

The following paragraphs will discuss how to synthesize flows for long time malicious and benign API-based application behaviors, and human behaviors.

1) Synthesis of Flows for Malicious and Benign API-based Applications

Our methodology of how to synthesize flows for automated malicious and benign API usage behaviors in a long period of time follows the logic stated below.

(1) To begin, collect flows generated by a single post or comment API operation. Benign applications will post or comment with benign content, while malicious applications will post or comment with malicious content. Save those flows into a file, then each file will include all flows generated by this single API operation.

(2) In a similar way, repeat multiple single API post or comment operations, and save those flows into multiple flow files.

(3) For each malicious and benign API behavior, we combine flow files generated by corresponding single API operations with different intervals separately into a large flow file by only changing each flow's start and end time, while keep all other attributes the same. Time intervals are generated based on which malicious timing pattern and benign timing pattern the benign or malicious API-based application follows.

Based on the above method, the large generated flow file is the synthesized flows for malicious and benign API behaviors in a long period of time, and we can generate flows for each malicious and benign API-based application based on the time span we defined. We design Algorithm 1 to synthesize flows for OSN API based applications in a predefined time span. We first introduce the following definitions, and the Algorithm 1 is followed by those definitions.

(1) Flows set M: For each single benign or malicious OSN application post or comment behavior, we repeat those single behaviors for a large amount of times, then collect flows generated by each behavior into a separate file. The set M consists all those files, and each file includes all flows generated by a single OSN application behavior.

(2) Possibility density function F: For a malicious or a benign OSN application, its behave time points follow a particular timing pattern, and F is the corresponding possibility density function for that timing pattern.

(3) timeCount: This parameter indicates the time span of the flows we are synthesizing for the application.

(4) Possibility[t]: The possibility density that an OSN application at time point t will do a post or comment behavior.

(5) Set R: all flows in set R are the synthesized flows for an OSN application in timeCount seconds.

Algorithm 1: Generation of flows for Malicious or Benign OSN applications

```
1:  Input: flows set M, a random behavior possibility density function F
2:  Initialize possibility density array Possibility[timeCount]
3:  Initialize timeCount as 24*60*60 seconds
4:  Initialize result set R as empty set
5:  FOR t in timeCount DO
6:      Possibility[t] := F(t)
7:  END FOR
8:  FOR t in timeCount DO
9:      Generated a random number rand which range in (0,1)
10:     IF rand < Possibility[t] THEN
11:         Select flows S generated by a single OSN behavior in M
12:         Record the start time of first flow in S as firstStart
13:         FOR f in S DO
14:             IF f is the first flow THEN
15:                 Compute  f's start time as t
16:                 Compute  f's end time as t + f's end time - firstStart
17:             ELSE
18:                 Compute f's start time as t + f's start time - firstStart
19:                 Compute f's end time as t + f's end time - firstStart
20:             END IF
21:         END FOR
22:     Add those updated flows in S to result set R
23:     END IF
24:  END FOR
25:  Output:  flows set R is synthesized flows for a malicious OSN application
```

2) Synthesis of Flows for Human OSN Behaviors

The method of synthesizing human OSN flows is a little different from synthesizing flows for malicious and benign API applications. For API-based applications, their behaviors mainly consist of different API-based post or comment behaviors. However, human operations on social networks are much different, so we collect user follows based on a single user session instead of a single operation on social networks. A single user session includes all activities from a user opening an OSN webpage to closing it, and the user may perform various operations on OSNs at each user session, as discussed in Chapter V. The user session time duration distribution and click event pattern follow the pattern summarized in the real Chinese social network RenRen (G Wang, 2017).

To generate flows for human behaviors in a long time period, we combine flows generated by different user sessions together, only changing each flow's start and end time. Our methodology of how to synthesize flows for human behaviors in a long time period follows the steps below.

(1) First, collect flows generated by a single user session, then save those flows into a file. Each file includes all flows generated by this single user session.

(2) In a similar way, save multiple flow files generated by multiple user sessions.

(3) Combine flow files generated by each user session with different Poisson intervals into a large flow file by only changing each flow's start and end time while keeping other attributes the same. Human beings' use of social network intervals follow the Poisson distribution, so we use Poisson intervals here.

In this way, the generated large flow file is the synthesis of flows for human behaviors in a long time period, and we can generate flows for human OSN behaviors based on our defined time span. Our methodology of how to synthesize flows for human behaviors in a long time period follows Algorithm 2. We first introduce the following definitions, and Algorithm 2 is followed by those definitions.

(1) Flows set H: Each human user behavior in a human user session is instructed by the finite state machine. We run the finite state machine a large amount of times, then collect flows for each human user session, store corresponding flows for each behavior into a separate file. The set H consists all those files, and each file includes all flows generated by a single human user session.

(2) Intervals[]: Two adjacent human user sessions may have various intervals, and those intervals can be modeled as various Poisson process, the Intervals[] stored the intervals generated by a Poisson process.

(3) timeCount: This parameter indicates the time span of the flows we are synthesizing for the human user.

(4) curTime: Current time of synthesizing flows compared with the whole time span.

(5) Set R: all flows in set R are the synthesized flows for a human user in timeCount seconds.

Algorithm 2: Generation of flows for OSN human behaviors

```
1:  Input: flows set H, a random user session intervals array Intervals[]
2:  Initialize curTime as 0, interval counter i as 0
3:  Initialize timeCount as 24*60*60 seconds
4:  Initialize result set R as empty set
5:  WHILE curTime + Interval[i] < timeCount DO
6:        Compute curTime as curTime + Interval[i]
7:        Select flows S generated by a single OSN user session behavior in H
8:        Record the start time of first flow in S as firstStart
9:        FOR f in S DO
10:             IF f is the first flow THEN
11:                  Compute  f's start time as curTime
12:                  Compute  f's end time as curTime + f's end time - firstStart
13:             ELSE
14:                  Compute  f's start time as curTime + f's start time - firstStart
15:                  Compute  f's end time as curTime + f's end time - firstStart
16:             END IF
17:       END FOR
18:       Add those updated flows in S to result set R
19:       Increase i as i + 1
20:       END IF
21:   END WHILE
22:   Output:  flows set R is synthesized flows for a human user in a day
```

3) How We Control Flow Attributes When Generating Synthetic Flows

Each flow includes several attributes, such as flow start time, flow end time, flow duration, source IP, source port, destination IP, destination port, TCP flag, and packets number in each flow. In this section, we will discuss what attributes of flow are controlled in our synthesized flows, and how we control different attributes for synthesized flows.

For both benign and malicious OSN applications, their behavior can be defined from two aspects. The first is to decide when OSN applications do post or comment behaviors, the second is to decide what content they post or comment each time. For malicious OSN applications, their post or comment time points are decided by five malicious timing patterns by varying all possible parameters in our proposed malicious pattern formulations. The spam content (e.g. post with malicious URL) that they post/comment is downloaded from a Twitter spam dataset. The malicious OSN applications may use 1 or multiple accounts to post/comment spam information. For benign OSN applications, their post or comment time points are decided by a benign Poisson pattern by varying all possible parameters in Poisson distribution. The benign content (e.g. post for weather warnings) that they post is crawled from online benign bot-controlled Twitter accounts.

We have summarized how we simulate malicious and benign OSN application behaviors. Now we will answer the question of how we control flow attributes for benign and malicious OSN application behaviors. To begin, for malicious OSN applications, five malicious timing patterns decide flows' start time and the behavior of "post or comment with spam content for 1 or multiple accounts" decides the flows' other features. After that, for benign OSN applications, 1 benign timing pattern decides the flows’ start time, and the behavior of "post or comment with benign content" decides the flows’ other features.

For human behaviors, we use 27 streamlines of human behavior chains based on different event occurrence sequences to simulate all possible operations during each user session. A user session includes all human activities, from a user opening an OSN webpage to closing it. In each user session, user behaviors are actually driven by a series of click events, and the click event interval distribution in a user session is decided by the click pattern summarized in the real Chinese social network RenRen (G Wang, 2017). Since a human user may visit a social network several times a day, a human user may generate several user sessions. The intervals between different user sessions follow the Poisson distribution.

For human behaviors, the timing pattern of "Poisson distribution for intervals between user sessions" and "click events’ distribution within a user session" work together to decide flows’ start time, and various human behavior streamlines decide flows’ other features.

To summarize, in Chapter V we introduce how we simulate benign API-based application behaviors, malicious API-based application behaviors and human behaviors. After that, to speed up data generation, we synthesize flows for long periods of time of benign and malicious API-based application behaviors and human OSN behaviors. We also analyze how we control flow-level features for different synthesized flows.

CHAPTER VI

DEEP LEARNING BASED APPROACH OF DETECTING FLOWS FROM MALICIOUS API-BASED APPLICATIONS

We have discussed how to generate flows for malicious API-based applications, benign API-based applications and human OSN behaviors. Our purpose is to detect flows from malicious API-based applications. In this chapter, we will introduce how to use our generated ground truth to train a Convolutional Neural Network (CNN) model. This chapter also includes model selection, ground truth labeling, dataset size, data preprocessing and model structure.

A. Model Selection

Convolutional Neural Network (CNN) has been a very popular tool to finish various machine learning tasks in recent years. It is particularly powerful to learn hierarchical level features from complex high dimension data automatically, then finish classification task effectively. The aggregated flows in our project are actually high dimension data, so CNN is a good tool to classify malicious API flows and other benign flows. CNN needs uniform input size, but each of the aggregated flows in our project contains a different number of flows. Therefore, we normalize each of the aggregated flows by transforming each of the aggregated flows into an image. Each image still reserves the main useful information of each aggregated flow group, and then normalized images are used as the input to train the CNN model.

B. Ground Truth Labeling

There are three OSN behaviors defined in this thesis: malicious API-based application behaviors, benign API-based application and human OSN behaviors. Our purpose is to detect flows of malicious API-based application behaviors separately from flows generated by all three behaviors. For flows generated by malicious API-based application behaviors, we label them as malicious flows. For flows generated by benign API-based application behaviors and human behaviors on social networks, we label them as benign flows.

C. Dataset Size

For flows generated by each malicious API-based application behaviors and benign behaviors, we aggregate their flows, and then convert each aggregated flow into an image, which can act as the normalized input of CNN model. We generate 40,000 aggregated flows for both benign behaviors and malicious API application behaviors. The data size of categorized aggregated flows for each behavior is shown in Table 2.

Table 2. Aggregated flows number for each OSN behaviors

Aggregated flow type	# of aggregated flows
Aggregated malicious OSN application flows	15,000
Aggregated benign OSN application flows	15,000
Aggregated human user operations flows	10,000

As we talked about in the data generation chapter, there are five malicious API-based applications. They post or comment with spam contents, and behavior time points decided by five malicious timing patterns. We represent malicious applications with different malicious API usage timing patterns as bad1, bad2, bad3, bad4, and bad5, and represent the benign API- based applications with benign timing pattern as good1, while human behaviors are labeled as good0. Table 3 has shown generated aggregated flows' size for each applications' malicious timing patterns and benign timing patterns. The data set is split into a training set and a test set. The training set has 80% of all labeled aggregated data, while the test set includes 20% of all labeled aggregated data. The 32,000 training set are used to train the CNN model, while we reserve 8,000 test data to evaluate the performance of the trained model.

Table 3. Aggregated flows number for applications with different malicious/benign application patterns

Aggregated flow pattern type	# of aggregated flows
Malicious application pattern bad1	3,000
Malicious application pattern bad2	3,000
Malicious application pattern bad3	3,000
Malicious application pattern bad4	3,000
Malicious application pattern bad5	3,000
Benign application pattern good0	15,000
Benign human user pattern good1	10,000

D. Flow Data Preprocessing

For flows generated by each application and human behaviors, we will preprocess flows first, then train a deep learning model using preprocessed flows instead of raw flows. The flow pre-processing process includes several steps: flow extraction, aggregation, and normalizing aggregated flows into images. Normalized images will be act as the input of the deep learning model.

First, we need to extract the OSN-generated flows. When we visit the small social network WordPress on the client side, the client is actually communicating with the server of our deployed small social network, and this process can generate a lot of network packets. The client is installed with flow data generation and collection software, so it can generate and collect flow data based on current incoming and outgoing packets on this client. However, when we simulate those three kinds of behavior on our client, the client is probably communicating with multiple network servers at the same time, so our collected flow data not only contains flows communicating with the WordPress server, but also includes flows generated by other applications on the client. Therefore, we need to extract those flow data, which are generated by communication traffic between this client and the WordPress server. To collect those flows generated by our synthesized behaviors, we extract flows whose source IPs or destination IPs belong to the WordPress server. Our IP address matching method can also extract flows generated by other OSNs, such as Twitter and Facebook.

After extracting WordPress traffic flows on the client, we will aggregate flow data. A single simple behavior on WordPress and any other website can generate several traffic flows, so a single traffic flow can carry very little information about how users actually operate on an OSN. Therefore, we want to aggregate traffic flows that occur in a relatively long pre-specified time window. The aggregated traffic flows can then carry information of how a user behaves in the pre-specified time window. Aggregated flows only collect flows that occurred in a specific time slot, and still preserve each flow's start time, end time and other attributes, so the timing and other features are still preserved in aggregated flows. If an API-based application is used with malicious purposes in this pre-specified time window, we can detect those aggregated traffic flows from this client.

Since aggregated flows have a different number of flows, and the CNN model needs uniform input size, we normalize the input of each aggregated flows by converting it to an image which carries the main useful information for the aggregated flows. Each aggregated flow group is converted into a scatter image, and each point in this scatter image carries the main information for a flow in this aggregated flow group. A flow is converted to a point in the image, so the image consists of all points converted by all flows in an aggregated flow group.

A point carries four main attributes information from the corresponding flow: flow start time, flow duration, packet number and TCP flag. Those four attributes information of a flow are converted to a point's location and (R, G, B) color value in the image. To be more specific, flow start time attribute decides a point's location (x, y) in this image: x axis is the flow's generated minute, while y axis is the flow's generated second. The flow duration attribute decides the point's R value, the packet number attribute decides the point's G value, while the TCP flag decides the point's B value. In this way, the four main useful attributes of a single flow data are carried by the corresponding point's four features in the scatter image. Each converted image can carry information for all flows with their four attributes' information in the aggregated flow group. In Figure 8, there are three examples of three converted images of aggregated flows from malicious API-based application behaviors, benign API-based application behaviors and human behaviors separately.

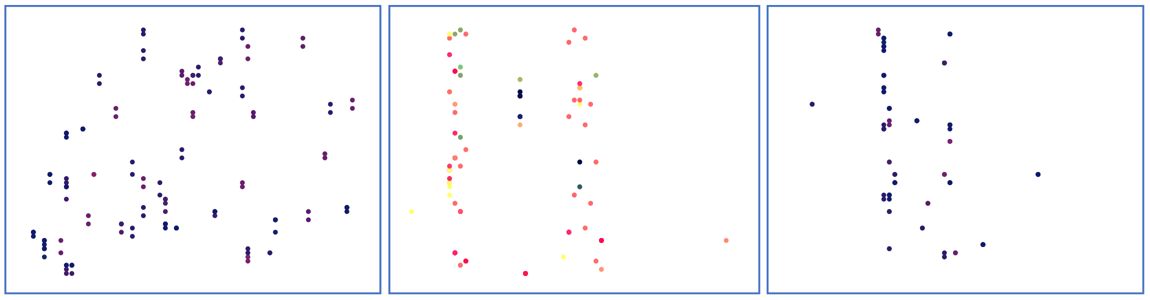

(a) Benign application flows (b) Human user flows (c) Malicious application flows

Figure 8. Converted images for human, malicious application and benign application flows

E. CNN Model Structure

By changing all related parameters and layers, the CNN model with structure in Figure 9 can achieve the best performance. Our trained CNN model consists of 10 layers. In the CNN model, the output of the previous layer is the input of the next layer, as seen in Figure 9. The first layer is the input of the model, and the last layer is the prediction result of the model. The input layer of the model takes 128*128*3 dimension matrixes as input, which are read from images converted from aggregated flows. For each input data, the CNN model can predict it with label 0 or label 1 at the output layer. If it is predicted with 0, it indicates that the flows are generated by benign API-based application behaviors or human behaviors on OSNs. If it's predicted with 1, it indicates that the flows are generated by malicious API-based application behaviors.

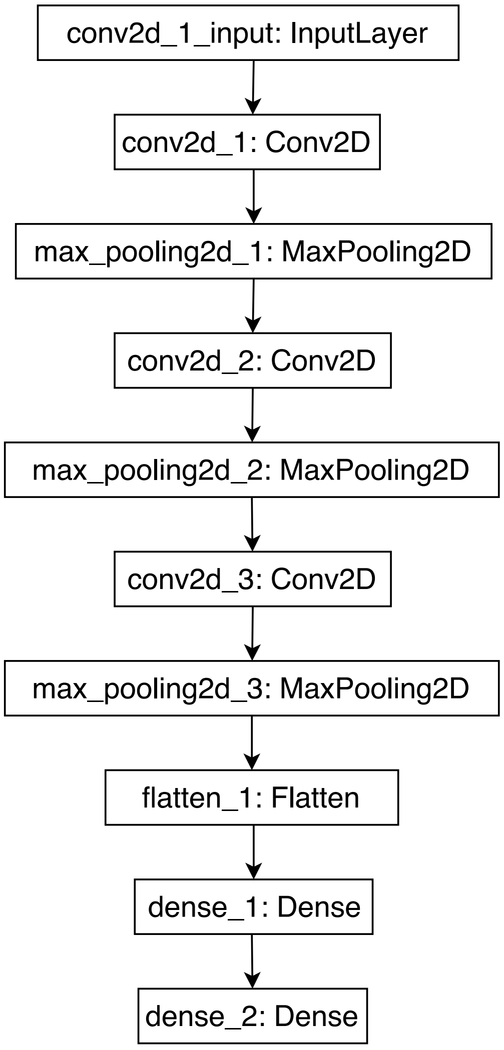

conv2d_1_input: InputLayer
conv2d_1: Conv2D
max_pooling2d_1: MaxPooling2D
conv2d_2: Conv2D
max_pooling2d_2: MaxPooling2D
conv2d_3: Conv2D
max_pooling2d_3: MaxPooling2D
flatten_1: Flatten
dense_1: Dense
dense_2: Dense

Figure 9. CNN model structure

To summarize, in this chapter, we describe how to label data, the preprocessing of flows into images to train the CNN model, and the training of the CNN model structure.

CHAPTER VII

EVALUATION

In this chapter, we evaluate the performance of our trained CNN model. Four metrics are used to evaluate the detection performance of the trained CNN model: accuracy, recall, precision and F1-measure. To begin with, we evaluate the overall detection result for all test set data. After that, we evaluate the detection performance when malicious API-based applications post or comment 10 times, 20 times, 30 times, 40 times, and 50 times. Afterwards, to obtain a better understanding of how our CNN model can detect each malicious API-based application's behaviors effectively, we evaluate CNN's detection performance when each malicious API-based application posts or comments 10 times, 20 times, 30 times, 40 times, and 50 times. At last, we evaluate the performance of the trained CNN model by detecting flows generated by three real world API-based malicious applications and three API-based benign applications.

A. Test Set Size and Evaluation Metrics

We reserve 8,000 (20% of generated ground truth data) aggregated flows as the test set. The test set includes aggregated flows for malicious API-based application behaviors, benign API-based application behaviors, and human behaviors on social networks. Their corresponding data sizes are shown in Table 4.

Table 4. Test set data size for each behavior

Test set aggregated flow type	# of aggregated flows
Malicious API-based applications	3,000
Benign API-based applications	3,000
Human OSN behaviors	2,000

We adopt the four most commonly-used metrics of accuracy, recall, precision and F1-measure to evaluate the performance of our trained CNN model. Accuracy is the proportion of predictions that are correct. Recall is the measurement of how many actual positive observations are predicted correctly. Precision measures how many positive predictions are actual positive observations. F1-measure is the harmony of precision and recall. We have TP, TN, FP and FN indicating the number of true positive, true negative, false positive and false negative in prediction results. Thus, all four metrics can be formulated based on TP, TN, FP and FN.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

$$F1 - measure = \frac{2TP}{2TP + FP + FN} \tag{9}$$

B. Overall Detection Performance for All Test Set

To begin with, we evaluate the overall performance of the trained CNN model by predicting the whole test set. As shown in Figure 10, the overall performance of our model is very good. It can achieve very high scores for accuracy, precision, recall and F1-measure at the same time.

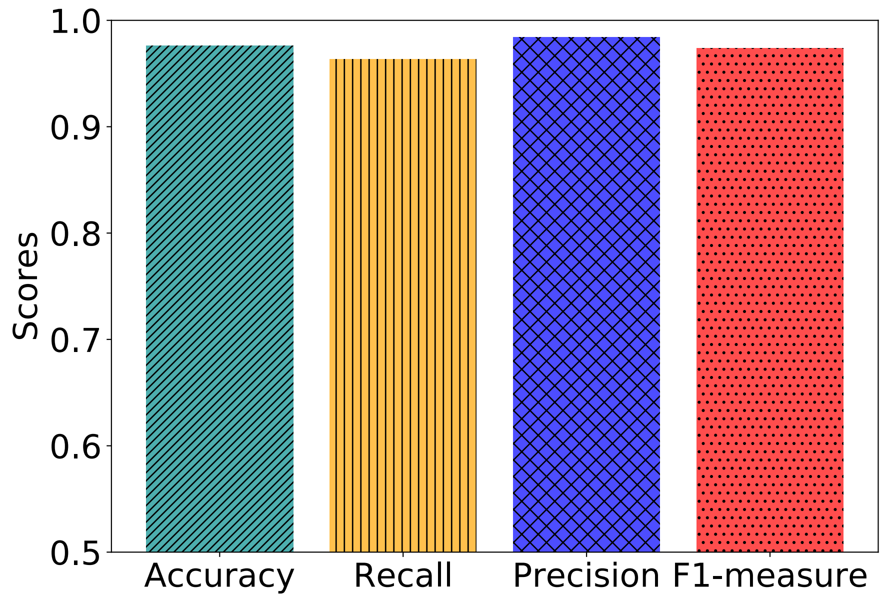


Figure 10. Detection performance for predicting all test sets

C. Detection Performance for Malicious API-based Applications with Different Post/Comment Frequencies

In part VII, it is shown that the trained CNN model's overall prediction performance is very good. In this part, we would like to check if when malicious API-based applications' post/comment times vary from 10 to 50 times, our trained model can detect those malicious API applications effectively.

In Figure 11, the x axis indicates malicious API applications post/comment total times in the aggregated flows, while the y axis shows our model's corresponding detection performance. Based on observation, we find that when malicious applications post/comment less frequently, our model's detection accuracy is relatively low. The accuracy is nearly 89.9% when malicious APIs post 10 times in a day. As malicious patterns post/comment increasingly frequently, our model's detection accuracy increases, and will reach up to nearly 99.3% when malicious APIs post 50 times.

When malicious API-based applications act more frequently, their malicious patterns become increasingly obvious, such that their flows become more easily detected by our model. Figure 12 compares two examples of the first malicious API application with different post/comment times. The Figure 12a shows when the first malicious API application posts only 10 times, and has a malicious pattern which is not that obvious. In Figure 12b, the malicious pattern is much more obvious when it posts 50 times than when it posts 10 times. Therefore, we can find when malicious API-based applications post times increase, their malicious patterns will become more obvious. Our method can detect flows from frequently-acting API-based applications more effectively.

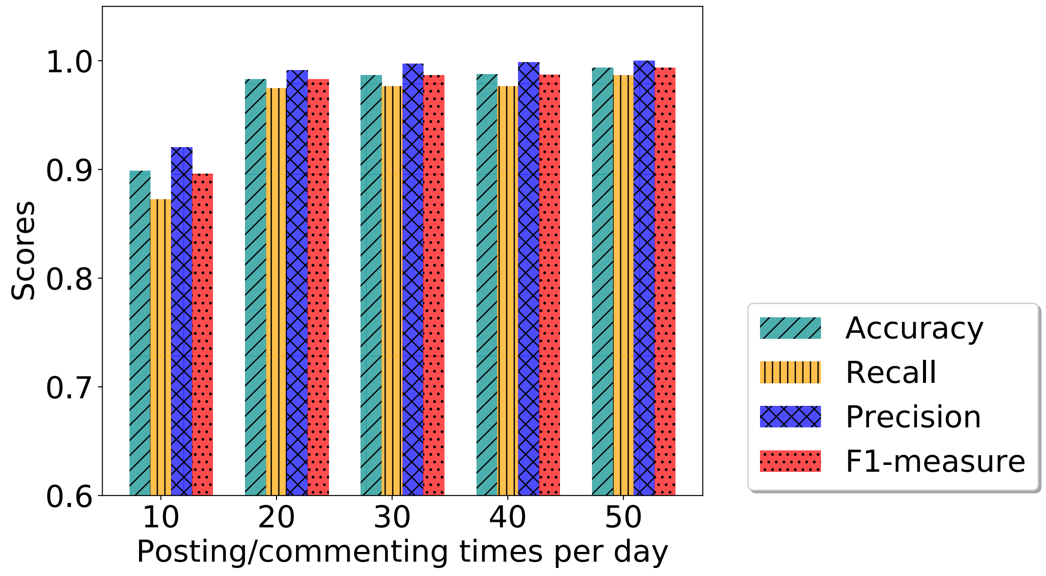
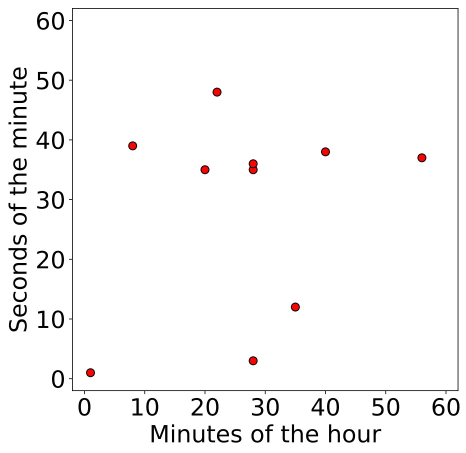
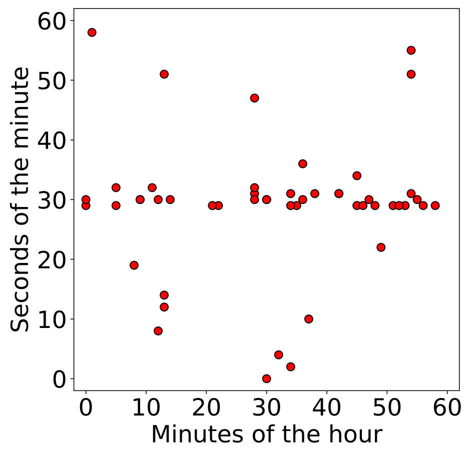


Figure 11. Detection performance for malicious API usage with different posting/commenting times per day



(a) 10 times



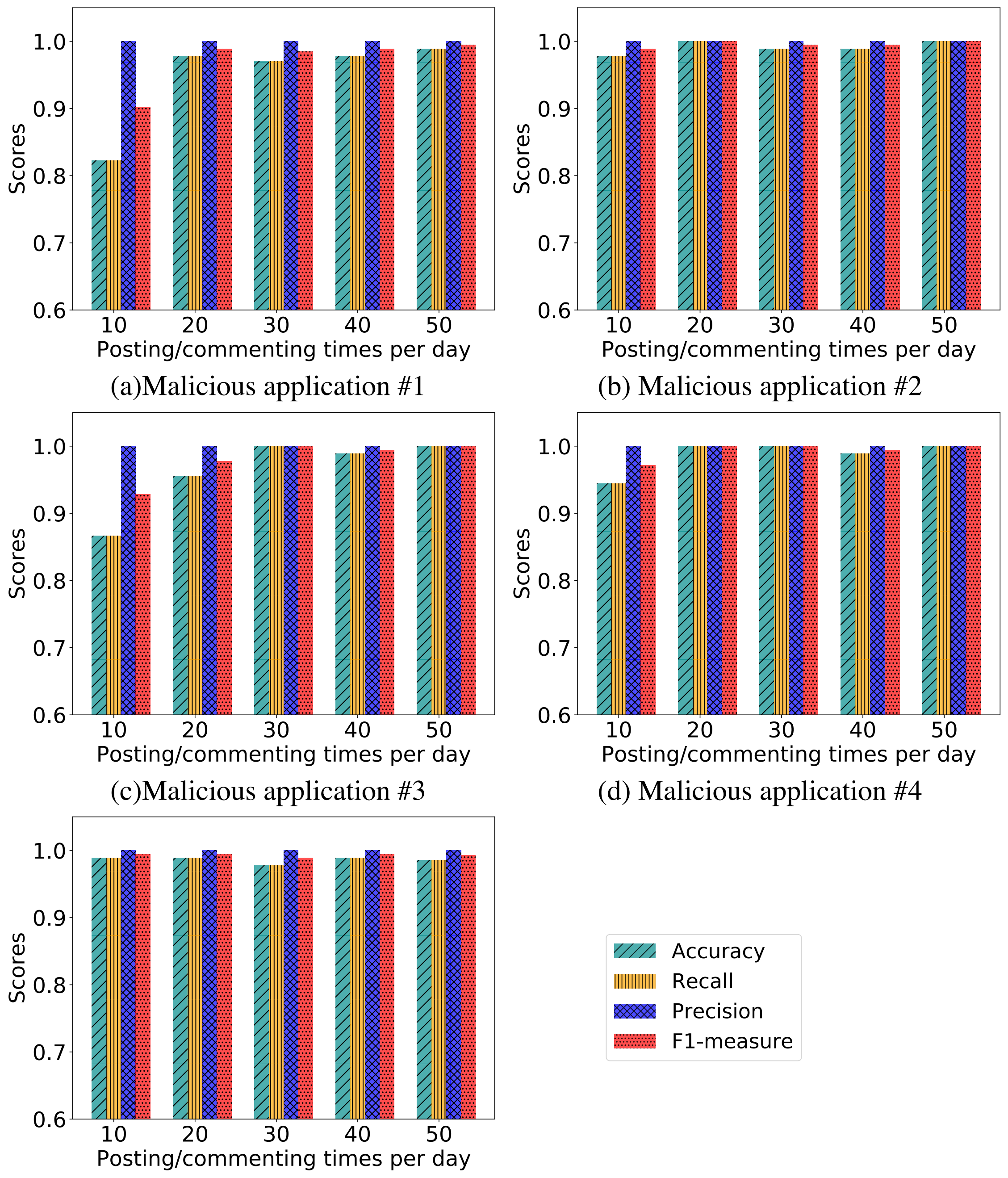
(b) 50 times

Figure 12. Comparison for malicious API application #1 posting/commenting 10 and 50 times, respectively

D. Detection Performance for Each Malicious API-based Application with Different Post/Comment Frequencies

To obtain a better understanding of how our trained model can detect each malicious API-based application's generated flows effectively when the application posts/comments times changes, we display another group of detection results for each malicious application with post/comment frequency changing from 10 times, 20 times, 30 times, 40 times to 50 times per day.

Figure 13 shows the performance of our trained model in detecting flows of each malicious API-based applications when their posting frequency changes from 10 times to 50 times per day. Based on our observation, we found that the detection result shows two patterns. For malicious API-based application bad1 and bad3, when post/comment times are less frequent, our model's detection accuracy is very low. When their posts/comments become more frequent, our model's detection accuracy increases until it is very high. However, for malicious API- based application bad2, bad4 and bad5, our model's detection accuracy is always very high.



(a)Malicious application #1 (b) Malicious application #2

(c)Malicious application #3 (d) Malicious application #4

(e) Malicious application #5

Figure 13. Detection results for each malicious API-based application posting/commenting 10 to 50 times per day

The reason why those results show two different patterns are related to how different malicious API-based applications post/comment. The example of bad1 and bad2 is shown in Figure 14. For bad1, post/comment behavior time has relatively long intervals. When it posts less frequently (e.g., 10 times per day) the bad1 pattern is not obvious, so our model's detection accuracy is low. For bad2, post/comment behaviors mainly focus on a short time period. Even if it posts less frequently (e.g., 10 times per day), its malicious pattern is still very obvious, so our model's detection accuracy for bad2 is always high.

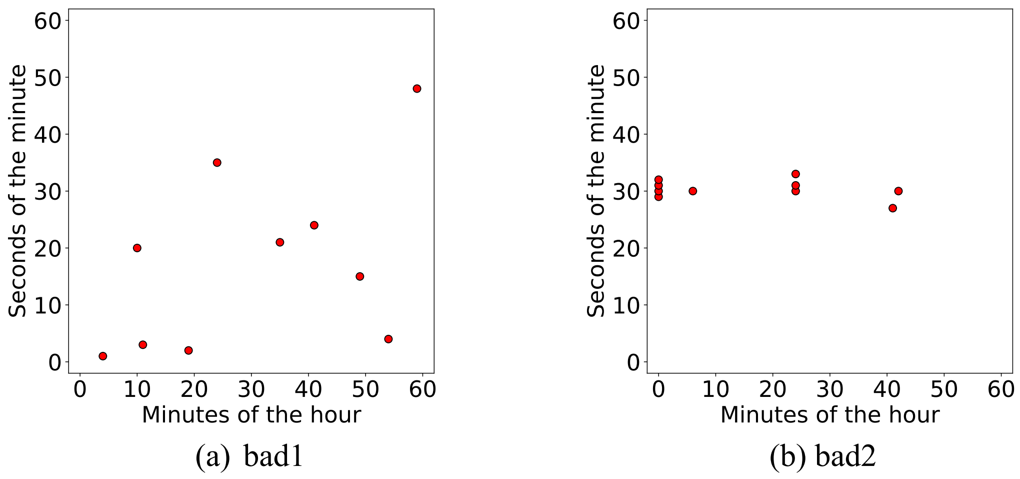


Figure 14. Comparison for bad1 application and bad2 application for posting/commenting 10 times per day

We also observed another phenomenon: when bad1 and bad3 post less frequently, even if our model's detection accuracy scores are low, the precision scores are still very high. Precision is a measurement of how many positive predictions are actual positive observations. We represent malicious API behaviors as positive, which is described in chapter VI. This high precision result indicates that our model's predicted malicious flows are very likely to be actual malicious flows, and our model may predict malicious flows as benign flows. This indicates when the behavior patterns of some malicious API-based applications are not obvious, our model can mistakenly predict those malicious flows as benign.

E. Detection Performance for Detecting Real World API-based Benign Applications and Malicious Applications

Since we have evaluated the performance of the trained model based on five synthesized malicious applications, we must demonstrate that our synthetic flows are very close to the real world generated malicious and benign flows. In this section, we use our trained CNN model to detect flows from three real world malicious OSN API-based applications and three real world benign OSN API-based applications.

The three real world benign OSN API-based applications are an earthquake bot providing real time earthquakes happening in specific locations, a news bot providing important news to people, and a weather warning bot reporting weather warning information. The three malicious OSN API-based applications are all spam applications.

We collected and aggregated corresponding flows by running those applications on our platform, and then used our trained model to detect aggregated flows generated by three malicious applications and three benign applications. Figure 16 shows our model's performance for detecting flows for malicious API-based application behaviors. As we can see, our detection model performs well and can detect malicious flows with accuracy as high as 99.7%, 98.8% and 99.1%. Figure 15 shows the performance for detecting flows for benign API-based application behaviors. Our detection model performs well and can detect benign application flows with accuracy as high as 93.4%, 91.1% and 99.2%. The precision score is 0, and recall and F1-measure cannot be calculated because of no instances of true positive (TP) and false negative (FN) results. All samples are either correctly predicted as TN or mistakenly predicted as FP. The Figure shows that our model can detect flows from real world benign and malicious API-based applications well.

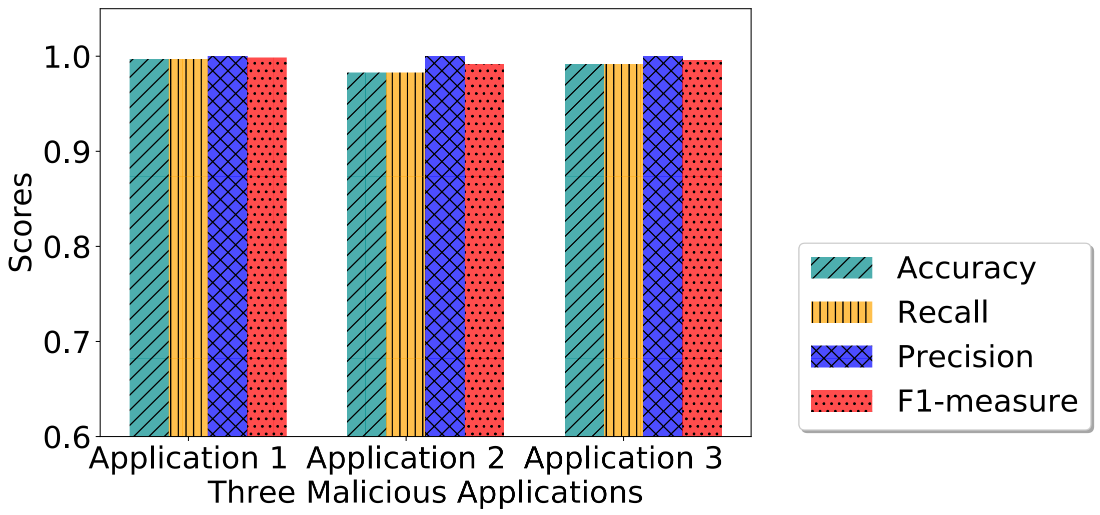


Figure 15. Detection results for flows generated by three real world malicious API-based OSN applications

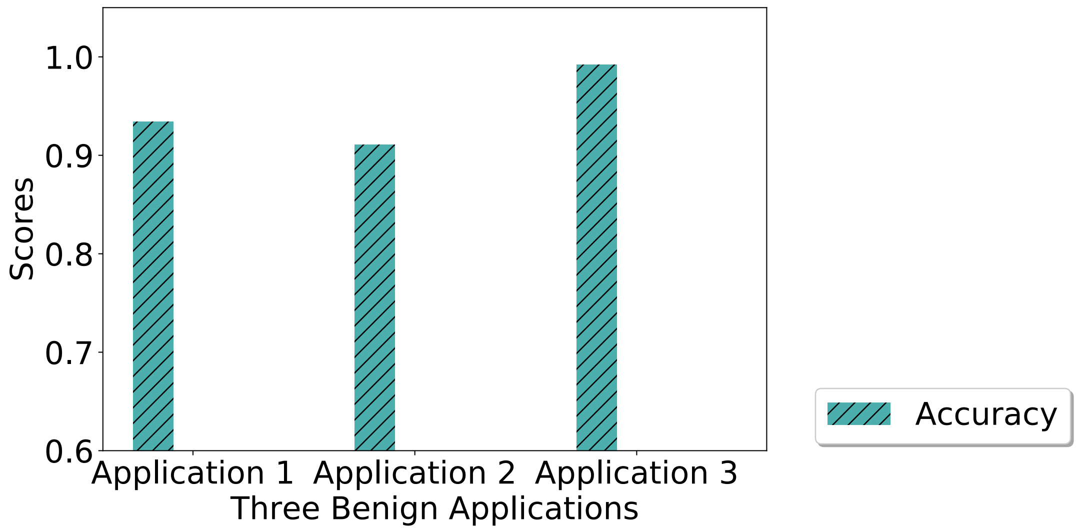


Figure 16. Detection results for flows generated by three real world benign API-based OSN applications

To summarize, our trained CNN model might be not able to identify flows for some malicious API-based applications very accurately when they only post/comment 10 times per day. When their post/comment times increase to at least 20 times per day, our model shows very good prediction performance. In addition, our model is also able to label flows generated by real world benign and malicious OSN API-based applications with good performance.

CHAPTER VIII

DISCUSSION

In this chapter, we are going to discuss some concerns and the future work for this research. The first concern is that some people are worried that a single timing pattern feature is enough to detect flows generated by malicious OSN applications, and therefore it is unnecessary to train a deep learning model based on other features. The second is that people may claim that our project can only detect flows from malicious OSN programs that demonstrate similar timing patterns as the five malicious timing patterns we used. For future studies, we are going to discuss the real-world applications of our work in helping NSP detect flows from malicious OSN applications.

A. Is A Single Timing Pattern Feature Enough to Detect Flows Generated by Malicious OSN Applications?

Malicious OSN applications have the malicious purpose of spreading spam information, so their behavior patterns are probably different from benign applications and human behaviors. It is known that malicious OSN applications' behavior timing patterns may be different from benign applications and human behaviors based on previous papers, and some people may think that a single timing pattern feature is sufficient to detect flows generated by malicious OSN applications. In this project, we train a deep learning model based on four features: flow start time, flow duration, packet number in each flow, and TCP flag in each flow. To investigate this concern, in this section, we train another CNN model only based on the timing feature, and show the detection result for the whole test set in Figure 17.

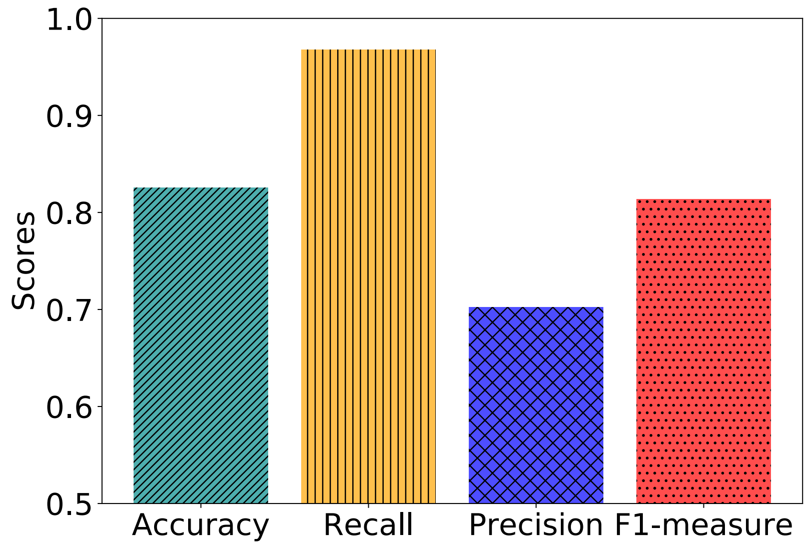


Figure 17. Detection result for predicting the test set for the model with only the timing feature

As we can see, the detection accuracy of the single timing-feature based CNN model can only reach to 82.5% percentage accuracy and 70.2% precision. The 70.2% precision indicates only 70.2% labeled malicious flows are actually maliciously-generated by malicious OSN applications, and many benign flows may be mislabeled as malicious. The detection performance of the single timing-feature based model is much worse than our trained four-feature based deep learning model. In our four-feature based deep learning model, the detection result for the whole test set achieved accuracy as high as 98.7% and precision as high as 97.5%.

To obtain a better understanding of why the timing feature-based model obtains overall lower accuracy and lower precision, we investigated four questions and trained four timing-feature based models to answer those questions: (1) Can we distinguish malicious application flows from benign application flows from just their timing features? Model 1: trained under the presence of only malicious and benign application flows (2) Can we distinguish malicious application flows from human flows from just their timing features? Model 2: trained under the presence of only malicious application flows and human flows (3) Can we distinguish benign application flows from human flows from just their timing features? Model 3: trained under the presence of only benign application flows and human flows (4) Can we distinguish benign application flows, malicious application flows, and human flows from just their timing features? Model 4: trained under the presence of benign application flows, malicious application, and human flows.

Table 5. Detection accuracy result for different flow combinations in each timing-based model

	Malicious application detection accuracy	Benign application detection accuracy	Human behavior detection accuracy
Model 1	96.0%	98.9%	-
Model 2	99.6%	-	75.8%
Model 3	0	99.6%	99.2%
Model 4	96.7%	98.9%	25.1%

In Table 5, the timing-feature based Model 1 is trained by malicious application flows and benign application flows. We find that Model 1 can label flows from benign applications with accuracy as high as 98.9%, and flows from malicious applications with accuracy as high as 96.0%. This result indicates that a single timing feature is enough to distinguish flows generated by malicious applications or benign applications. For timing-feature based Model 2 which is trained by flows from malicious applications and human behaviors, we find that our trained Model 2 can label malicious application flows with an accuracy of 99.6%, while label flows from human behavior only with an accuracy of 75.8%. This indicates that a single timing pattern is not enough to distinguish flows generated by malicious applications or human behaviors. For Model 3 trained by benign application flows and human flows, the detection performance for labeling benign application flows and human flows is very good, and a single timing feature is enough to distinguish flows generated by benign applications or human behaviors.

For timing-feature based Model 4 trained by human flows, benign application flows and malicious application flows, we find that the model can distinguish benign and malicious application flows with very good performance, but detection of human flows resulted in very low accuracy. Based on Model 1, we know the timing feature is enough to distinguish benign application flows or malicious application flows. From Model 3, we know we know the timing feature is enough to distinguish benign application flows or human flows. In Model 2, human flows can be mislabeled as malicious flows when only detecting with the timing feature. Model 4 resulted in low accuracy in detecting human flows, and this is also caused by some human flows being mislabeled as malicious application flows (we have checked the detection result in labeling human flows, and many human flows are indeed mislabeled as malicious in Model 4), which is the same as Model 2. In Model 2, the model has already had a very hard time distinguishing human flows from malicious application flows only based on the timing feature. It can be understandable for Model 4 to have bad performance when distinguishing human flows from benign and malicious application flows too.

We can conclude based on Table 5 that the timing feature is not sufficient for distinguishing between human and malicious applications, but can help distinguish malicious from benign applications and benign applications from humans. The timing feature is an important feature for detecting malicious flows, but a single timing feature is not enough to solely detect flows from malicious applications, because it can mislabel human flows as malicious, which will result in a high false positive rate.

We have already found that the timing-based Model 4's low detection accuracy and precision is mainly caused by mislabeling some human flows as malicious, so the single timing pattern is not enough to distinguish flows generated by malicious applications or human behaviors. In this situation, we would like to investigate whether other features help distinguish between human flows or malicious application flows. Which features are particularly helpful? Does a combination of all useful features further improve the detection result?

In our project, the malicious application flows' detection model is trained based on timing pattern and three other features: TCP flag, packet number in each flow, and flow duration. We represent feature TCP flag, packet number in each flow, flow duration as feature 1, feature 2, feature 3 respectively. To check whether other three features (feature1, feature 2, and feature 3) are helpful to distinguish flows generated by human behaviors or malicious OSN applications, we train another 4 models with the timing feature and another feature: (1) Model 5: trained based on timing and feature 1 to distinguish human flows or malicious application flows. (2) Model 6: trained based on timing and feature 2 to distinguish human flows or malicious application flows. (3) Model 7: trained based on timing and feature 3 to distinguish human flows or malicious application flows. (4) Model 8: trained based on timing and feature 1,2,3 to distinguish human flows or malicious application flows.

When distinguishing human flows and malicious application flows by training a model only based on timing pattern, the model's accuracy for detecting human flows is 75.6% in Model 2. If we train the model with timing and one of the other three features, the detection accuracy for human flows can be improved, as is shown in Table 6. If feature 2 is added with the timing pattern to train Model 6, the accuracy of labeling human flows can be increased to an accuracy of 92.4%. If feature 1 is added to Model 5, the accuracy will increase to 82.8%. Feature 3 is the least useful feature, as it can only help improve accuracy to 79.9% in Model 7. Model 8 is trained based on timing pattern and all the other three features. This four-feature based Model 8 can detect the human flow detection with an accuracy as high as 99.8%. This result indicates that all the other three features can help distinguish human flows from malicious application flows independently.

Table 6. Detection accuracy result for different flows in different feature based models

	Human flow detection accuracy	Malicious application detection accuracy
Model 5: timing + feature 1	82.8%	99.5%
Model 6: timing + feature 2	92.4%	99.4%
Model 7: timing + feature 3	79.9%	99.1%
Model 8: timing + feature 1,2,3	99.8%	100.0%

A combination of the three other features improves accuracy for distinguishing human flows and malicious applications, with the best detection accuracy for human flows.

Based on the above analysis, when Model 2 is trained under the presence of only malicious and benign application flows, it can’t achieve good performance when it’s only trained on a single timing pattern. If we train it with timing and all other three features in Model 8, the detection performance is very good.

We also train a model with timing and three other features to distinguish flows between malicious or benign applications in Model 9 in Table 7, where we observe that Model 9’s detection performance is also very good. Model 10 is trained by timing and the other three features to distinguish human flows and benign application flows, resulting in very good detection performance.

Table 7. Detection accuracy result for different in each four-feature based model

	Malicious application detection accuracy	Benign application detection accuracy	Human behavior detection accuracy
Model 8	99.8%	-	100.0%
Model 9	97.9%	96.6%	-
Model 10	0	100.0%	100.0%
Model 11	97.0%	98.8%	99.5%

Therefore, if we train models based on timing and the other three features, the detection performance will not hurt for distinguishing between human flows and benign applications, and will not hurt for distinguishing between malicious application flows and benign applications flows.

Finally, Model 11 is trained based on timing and the other three features to distinguish malicious application flows, benign application flows and human flows. The detection performance for all flows in this Model is also very good. Therefore, a timing feature is not enough to distinguish all three flows, but timing in addition to the other three features are enough.

B. Can Our Project Only Detect Flows from Malicious OSN Programs That Demonstrate A Similar Timing Patterns with Our five Used Malicious Timing Patterns?

We used four features to detect flows from malicious applications in this project: flow start time, TCP flag, flow duration, packet number in each flow. As we can see in chapter VIII, the timing feature could be an important feature for us to detect flows generated by malicious OSN applications. However, to achieve a high detection precision result, the other three features are also helpful. In a more accurate way, this project is to identify applications that exhibit both five malicious timing patterns and other flow-level malicious features, instead of identifying applications that only exhibit the five malicious timing patterns.

In this project, we have five malicious timing patterns decide when malicious OSN applications post/comment with spam content, so our trained model is indeed supposed to detect flows from malicious applications showing similar timing patterns with our used five malicious timing patterns.

In order to create an expressive set of malicious timing patterns, we extended the findings of paper (CM Zhang, 2011). They present five malicious timing patterns of real-life spam accounts. These timing patterns are a good basis to describe possible malicious timing behaviors, but they are not complete. For example, the paper suggests that posting once a minute throughout a day is an example of malicious behavior, but one can also further infer that posting twice a minute is also a reasonable example of malicious behavior. Therefore, we create an extensive set of possible malicious timing behaviors through slight modifications of the timing behaviors presented by this paper. In order to extend one of the five malicious timing patterns presented in that paper, we first create a model that describes the presented malicious timing pattern, and then we further add variances to this model, which creates an extensive set of possible malicious temporal patterns.

In particular, the parameters of our model are each given a range of realistic values, and each derived malicious temporal pattern is a specific instance of possible parameter values of our model. This process creates a comprehensive set of malicious timing behaviors. Based on this comprehensive set, we trained a CNN to classify future flows that exhibit similar timing patterns as malicious. In fact, we found and downloaded three malicious spam programs, and for each downloaded program, a specific instance of possible parameter values of our model describes its timing pattern. Therefore, it is no surprise that our CNN successfully detected the flows generated by these spam programs as malicious.

To summarize, we varied all related parameters to simulate all possible malicious timing pattern instances, and our simulated instances can cover a wide variety of timing patterns used by real-world malicious programs. If the real world malicious program's behavior timing pattern is covered in our dataset, our trained model can detect this malicious application's generated flows.

CHAPTER IX

CONCLUSION

While most social network providers release some APIs for third-party developers to integrate OSN services to their own software, these APIs can be misused widely by malicious OSN applications, causing security, privacy and liability concerns to OSN providers, network service providers (NSPs), and users. This thesis mainly studies how NSPs may apply a methodology which first preprocesses network flows and then converts useful flow level features to images as input to train a deep learning model to detect network flows from malicious API- based OSN applications. The evaluation results show that via this methodology, we can detect flows generated by malicious OSN applications with 97.6% accuracy and only 1.6% false positive.

CHAPTER X

FUTURE WORK

We have proposed that our project can be used to detect flows generated by malicious OSN API-based applications for NSPs. In this chapter, we will discuss how to use our work help NSPs detect flows generated by malicious OSN applications in reality.

To make use of our work in the real world, NSPs should deploy an OSN flow collector running continuously at its border router, where all incoming and outgoing traffic can be caught in this network. This process should not require that much effort, because flow data only aggregates packet head information, and the flow data size is much smaller than the total packets' size. If a flow caught by the NSP has a source IP address or destination IP address belonging to an OSN, then this flow is generated by a connection between a machine inside the NSP and an OSN. There are real time online router tables that can provide IP blocks for an OSN by sending requests to BGPstream (O Chiara, 2016), so the NSP can decide whether a flow's source or destination address belongs to an OSN through IP prefix matching. In this way, NSPs can obtain all traffic flows generated between machines inside its network and an OSN.

The flow data are aggregated packet headers. When NSPs collect all OSN flows for each machine (associated with an IP) inside the network, NSPs can use our proposed method to detect whether flows generated between an IP and OSN servers are malicious. If flows are detected as malicious, it indicates that the machine with this IP is running malicious OSN applications. In response, the NSP can decide to block the bad traffic, block this compromised IP, or just not do anything.

To use our proposed method to train a deep learning model for detecting malicious flows, NSPs need to obtain the ground truth by labeling malicious OSN application flows, benign OSN application flows and human OSN flows for a particular OSN, then train the malicious flow detection model based on the ground truth for this OSN. Future work may further include automating the training of this model.

REFERENCES CITED

Almaatouq, A., Shmueli, E., Nouh, M., Alabdulkareem, A., Singh, V.K., Alsaleh, M., Alarifi, A. and Alfaris, A., 2016. If it looks like a spammer and behaves like a spammer, it must be a spammer: analysis and detection of microblogging spam accounts. *International Journal of Information Security*, 15(5), pp.475-491.

Ahmadinejad, S.H. and Fong, P.W., 2013, May. On the feasibility of inference attacks by third-party extensions to social network systems. In Proceedings of the *8th ACM SIGSAC symposium on Information, computer and communications security* (pp. 161-166). *ACM*.

Barford, P. and Plonka, D., 2001, November. Characteristics of network traffic flow anomalies. *In Internet Measurement Workshop* (pp. 69-73).

Benevenuto, F., Magno, G., Rodrigues, T. and Almeida, V., 2010, July. Detecting spammers on twitter. *In Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)* (Vol. 6, No. 2010, p. 12).

Benevenuto, F., Rodrigues, T., Cha, M. and Almeida, V., 2012. Characterizing user navigation and interactions in online social networks. *Information Sciences*, 195, pp.1-24.

Claise, B. (2004). Cisco systems netflow services export version 9.

Fazil, M. and Abulaish, M., 2018. A hybrid approach for detecting automated spammers in twitter. *IEEE Transactions on Information Forensics and Security*, 13(11), pp.2707-2719.

François, J., Wang, S. and Engel, T., 2011, May. BotTrack: tracking botnets using NetFlow and PageRank. In *International Conference on Research in Networking* (pp. 1-14). Springer, Berlin, Heidelberg.

Herrera-Joancomartí, J. and Pérez-Sola, C., 2011, July. Online social honeynets: trapping web crawlers in OSN. In *International Conference on Modeling Decisions for Artificial Intelligence* (pp. 1-16). Springer, Berlin, Heidelberg.

Hofstede, R., Bartoš, V., Sperotto, A. and Pras, A., 2013, October. Towards real-time intrusion detection for NetFlow and IPFIX. In Proceedings of the 9th *International Conference on Network and Service Management (CNSM 2013)* (pp. 227-234). IEEE.

Kind, A., Stoecklin, M.P. and Dimitropoulos, X., 2009. Histogram-based traffic anomaly detection. *IEEE Transactions on Network and Service Management*, 6(2), pp.110-121.

Móczár, Z. and Molnár, S., 2011, September. Comparative traffic analysis study of popular applications. In Meeting of *the European Network of Universities and Companies in Information and Communication Engineering* (pp. 124-133). Springer, Berlin, Heidelberg.

Mondal, M., Viswanath, B., Clement, A., Druschel, P., Gummadi, K.P., Mislove, A. and Post, A., 2012, December. Defending against large-scale crawls in online social networks. In Proceedings of the *8th international conference on Emerging networking experiments and technologies* (pp. 325-336). *ACM*.

Orsini, C., King, A., Giordano, D., Giotsas, V. and Dainotti, A., 2016, November. BGPStream: a software framework for live and historical BGP data analysis. In Proceedings of *the 2016 Internet Measurement Conference* (pp. 429-444). *ACM*.

Rosenberg, M., Confessore, N., and Cadwalladr, C., "How Trump consultants exploited the facebook data of millions," *New York Times*. [Online] Available: https://www.nytimes.com/2018/03/17/us/ politics/cambridge-analytica-trump-campaign.html

Saroop, A. and Karnik, A., 2011, December. Crawlers for social networks & structural analysis of Twitter. In *2011 IEEE 5th International Conference on Internet Multimedia Systems Architecture and Application* (pp. 1-8). *IEEE*.

Singh, R., Kumar, H. and Singla, R.K., 2012, December. Traffic analysis of campus network for classification of broadcast data. In 47th *Annual National Convention of Computer Society of India*. Int. *Conf. on Intelligent Infrastructure, MacGraw Hill Professional* (pp. 163-166).

Schneider, F., Feldmann, A., Krishnamurthy, B. and Willinger, W., 2009, November. Understanding online social network usage from a network perspective. In Proceedings of *the 9th ACM SIGCOMM conference on Internet measurement* (pp. 35-48). *ACM*.

van der Steeg, D., Hofstede, R., Sperotto, A. and Pras, A., 2015, May. Real-time DDoS attack detection for Cisco IOS using NetFlow. In 2015 *IFIP/IEEE International Symposium on Integrated Network Management (IM)* (pp. 972-977). *IEEE*.

Wang, A.H., 2010, July. Don't follow me: Spam detection in twitter. In 2010 *international conference on security and cryptography (SECRYPT)* (pp. 1-10). *IEEE*.

Wang, G., Zhang, X., Tang, S., Wilson, C., Zheng, H. and Zhao, B.Y., 2017. Clickstream user behavior models. *ACM Transactions on the Web (TWEB),* 11(4), p.21.

Wongyai, W. and Charoenwatana, L., 2012, May. Examining the network traffic of facebook homepage retrieval: An end user perspective. In 2012 *Ninth International Conference on Computer Science and Software Engineering (JCSSE)* (pp. 77-81). *IEEE*.

Zhang, C.M. and Paxson, V., 2011, March. Detecting and analyzing automated activity on twitter. In *International Conference on Passive and Active Network Measurement* (pp. 102-111). Springer, Berlin, Heidelberg.

Zheng, X., Zeng, Z., Chen, Z., Yu, Y. and Rong, C., 2015. Detecting spammers on social networks. *Neurocomputing*, 159, pp.27-34.

Zhenqi, W. and Xinyu, W., 2008, December. Netflow based intrusion detection system. In 2008 *International conference on multimedia and information technology* (pp. 825-828). IEEE.