Detecting Malicious Usage of Online Social Network APIs from Network Flows

Dan Li
dli@cs.uoregon.edu
Computer and Information Science
University of Oregon

***Abstract*—While online social networks (OSNs) provided 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 paper, we study how a network service provider may inspect its network traffic to detect network flows from malicious OSN applications. In particular, we implement a deep learning model to train and classify NetFlows that are generated by emulated human OSN behaviors, benign OSN applications, and malicious OSN applications on a testbed. We show that our solution is effective and can accurately label 97.6% NetFlows from the malicious OSN applications, with only 1.6% false positives.

***Index Terms*—OSN; NetFlow; network flow; malicious OSN application; OSN application; OSN API**

I. Introduction

Online social networks (OSN) have become extremely popular with an ever-growing user base. At the time of the writing, Facebook, Twitter, and WeChat each has 2.2 billion, 0.4 billion, and 0.5 billion users, respectively. In particuar, in order to further enrich and improve user experience, OSNs have provided public Application Programming Interfaces (APIs) to enable the development of OSN applications that 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. For example, by using OSN APIs, a malicous 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 [1].

OSN providers 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, such as, detect OSN spam accounts [2], [3], [4], [5], limit large-scale crawling activities [6], or detect malicious third-party OSN applications [7]. Even though OSN providers can detect misbehavior, there are still a lot of spam posts/comments posted from OSN API-based applications [8]. OSN providers cannot detect those spam attacks originating from malicious OSN applications effectively, or they do not want to detect them. Malicious OSN applications still can cause various security problems.

Those malicious OSN application can be running on the devices for OSN users throughout the Internet, causing security, privacy, and liability concerns to the network service providers (NSPs) for OSN users. 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 detect malicious OSN applications by monitoring API calls, or analyzing full knowledge of user profiles, posts and the entire OSN graph, but an NSP only has limited knowledge of OSN data (such as user posts, profiles, social behaviors, OSN graphs). An NSP can only access the traffic across its network, thus not able to leverage the aforementioned existing work toward malicious OSN applications. We therefore study how an NSP may monitor its traffic to detect traffic flows from malicious OSN applications, then the NSP knows better about what's happening inside its network, and this gives NSPs a choice in how to respond.

We make the following contributions:

1) We define a problem of detecting flows from malicious OSN API applications, whereas the flow data we use does not include traffic payload. In another words, our problem assumes no knowledge of OSN topologies or specific user profiles and data.
2) We propose a solution to detect flows from malicious OSN API applications. We define three flows in this project: flows from malicious OSN API applications, flows from benign OSN API applications and flows from human OSN behaviors. Our solution aggregates above three category flows separately, then normalizes each aggregated flows as the input to train a deep learning model, which can learn features effectively from high-dimensional network flows. For each machine in a NSP, we extract and aggregate flows between this machine

and an OSN, then normalize aggregated flows and use this normalized flows as the input of our previous trained deep learning model, then the trained model can label whether this normalized aggregated flows is generated by a malicious OSN API application running on a machine.

3) We implement our proposed solution on a test bed, where we simulate and collect flows for five malicious API-based applications, benign API-based applications and human OSN behaviors. To make collected flows cover as much malicious / benign flows as possible, we change parameters to run each applications or simulate human OSN behaviors by changing activity categories and time intervals. The trained deep learning model is able to detect flows generated by five malicious API-based applications with high accuracy and low false positive. In particular, the trained model is also able to label flows from three real world benign API-based applications and three real world malicious API-based applications. Our research demonstrates that it is feasible to detect flows from malicious API-based applications on OSNs. What's more, our proposed solution can be apply to any other social networks e.g. Facebook, Twitter.

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

II. Background and Related Work

Previous work related to this paper involves multiple topics. We organize them mainly from two aspects . The first aspect is how related work uses flow data to detect network attacks or anomalies, or uses network traffic data to study OSNs. Since most of current malicious API-based applications are used by spam/bot accounts to spread spam contents, or by crawlers crawling OSN data, the second aspect is how related work detects malicious spam accounts on OSNs, or prevent crawlers crawling OSN data.

To begin with, there are some related works that use flow data to detect network attacks/anomalies or study social networks by analyzing traffic data. Papers [8], [9], [10], [11], [12], [13], and [14] explore how people use flow data to detect network attacks or anomalies. Paper [8] uses campus traffic flows as testbed to detect anomaly broadcast traffic, while [9] extends the popular linkage algorithm PageRank to detect botnet traffic. The work [10] and [11] both detect network traffic flow anomalies by detecting flow-level anomaly features. Since it's feasible to detect network attacks and anomalies by analyzing network flow traffic, papers [12],[13], and [14] propose several real time intrusion detection systems based on monitoring network flow traffic. Those attacks or anomalies detection papers follow a similar idea: detect a specific attack by detecting this attack's flow-level features. However, different attacks have different flow-level features, so one attack detection method probably can not be applied to detect another attack with different flow-level features. Those specific attacks' features are different with OSN attacks' features, so their feature-based detection methods are probably not effective in detecting OSN attacks. We should find a method that can detect OSN attack features effectively.

Related work [15], [16], [17] and [18] introduces how people use traffic analysis methods to study social networks. These related research mainly focus on how to get a better understanding of how users use or interact with online social networks. Most network traffic analyzed in those work includes packet payloads, which carry much more information than our used network flow data, because flow data only carries aggregated packet header information. To sum up, both the research analysis data and traffic analysis purposes in those work are completely different from ours, so their research methodologies can hardly be applied to solve our problem.

APIs released by OSNs are supposed to work for third party developers to access OSN services, but they are widely misused by crawlers crawling OSN data or spammers spreading fraud or spam contents based on [19], [20] and [21]. Paper [19] crawls a large amount of sensitive OSN data by using OSN exposed APIs, and [20] even designs an API-based crawler which enables attackers crawling large amount of Twitter network structure level information. At the same time, malicious API application controlled spammer accounts are very common on OSNs, and [21] points out that many automated spam accounts on OSNs prefer use API rather than a web browser to spread fraud or spam contents.

From another aspect, there are also some works that detect malicious automated spam accounts or prevent crawling activities on OSNs.application In [22] and [6], the authors propose countermeasures to prevent attackers crawling sensitive OSN user data. Paper [22] proposes an "Online Social Honeynet" 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 deploying it in the real world. Paper [6] proposes a Genie system which can be deployed at OSN provider sides to thwart crawlers by detecting their different browsing patterns. This work need to analyze each user's trace of visiting their friends and non-friends, and this information is sensitive and not accessible from flow data, so their methodology can hardly be applied to solve our problem.

Recent works [2], [3], [4] and [5] introduce how people detect spam accounts on OSNs. In those papers, their spam account detection work is mainly conducted by following this logic: analyze spammer accounts' post contents, user profiles or user social behaviors, then detect those accounts by detecting those features. However, those features are not accessible in flow data, so we could not use their method to detect flows generated by malicious API-based applications on OSNs.

To sum up, there is no work detecting malicious flows from malicious API-based applications on OSNs. We are the first to detect malicious API-based applications generated for NSPs. The flow data used does not carry any sensitive user data, and our research demonstrates that it is feasible to detect flows generated by malicious API applications for OSNs.

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 aggregate flow for above three category flows separately, and normalize each aggregated flows, then use 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 it can label flows from malicious API-based applications. We implement 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 flows for malicious API-based applications and other benign social network flows. We deploy a small social network WordPress as test bed where users can browse and reply to each others' posts. We then collect flows for human OSN behaviors, benign API-based applications, and malicious API-based applications on this small social network. We perform 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 various and a large amount of human OSN behaviors, we write scripts to emulate various possible human behaviors on a social network website, and those operations include login, post, comment, browse behaviors, and so on. Human behaviors on social networks are driven by a series of click events, we use script to emulate those click events instead, and flows generated by script controlled click events should be same with flows generated by human controlled click events. For benign API-based applications, OSN APIs should only be a tool used by human beings, they 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, and human behave patterns can be modeled as different Poisson process [21], so benign API-based application behavior time points can 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, and at each behavior time points, the benign OSN application will post some useful contents. For malicious API-based applications, we write 5 known malicious application scripts, and their behavior time points are decide by 5 known malicious API-based application behave patterns [21]. Each application posts spam or malicious posts/comments following one of 5 known malicious timing pattern. We model each malicious API-based application behave 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/malicious API-based applications and human OSN behaviors, we collect corresponding flows when we simulate those three behaviors on the client side.

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

We've generated flows for above three behaviors. After that, we extract all flows generated by the communication between the client and our small social network by extracting those 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 only carries very little information about how user behaves on a OSN. Therefore, we aggregate flows that occurred in a pre-define time window together, and those aggregated flows can carry more information about how user behaves in pre-defined time window. For each aggregated flow group, we transform it into an image which carries main information of the aggregated flow group. Those transformed images are acted as the input to train deep learning model, and the deep learning model 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 those 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 build a small social network Wordpress blog system as a testbed, and implement our proposed solution to detect flows from malicious API-based applications on this testbed. What's more, our proposed methodology not only works on the deployed testbed, but also can be applied on any other OSNs, such as Twitter, Facebook, Instagram to detect flows from any malicious API-based applications for any NSPs.

IV. Problem Formulation

Now that we have introduced the background and motivation for this project, and given an overview of our work in Section III, we now provide a more specific problem formulation. In this section, to begin with, we introduce network flow data. After that, we explain how we define malicious API-based applications, benign API-based applications and human OSN behaviors, and 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 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 TCP protocol is used, the flow may also include TCP flag as its attribute. Flow attributes can also statical numbers for total bytes and total packets received by each flow. A flow data can have any attributes which can be read from packet headers. Figure 1 shows several examples of flow data.

Based on our description, the flow data can carry very limited sensitive social network user data, and it could be a little challenging to detect malicious social network flows for API-based applications. 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

Fig. 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/benign OSN applications, human communicate 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 NSP. NSPs provide network services to their users, and a NSP could be part of an Internet Services Provide (ISP). For example, a company network is a NSP, because it can provide network services all to workers in its company.

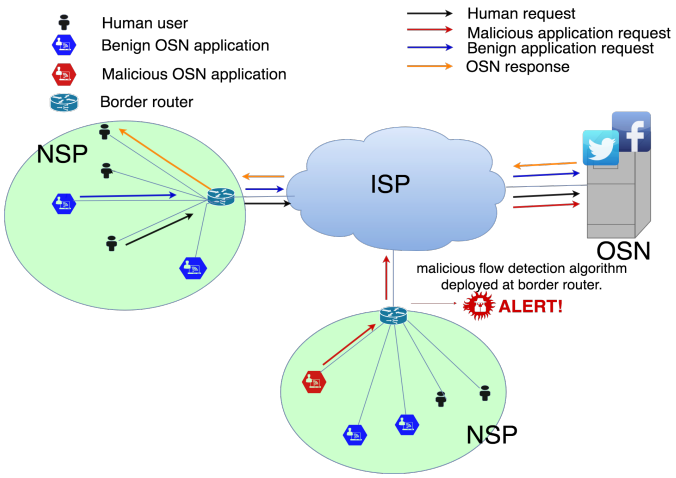


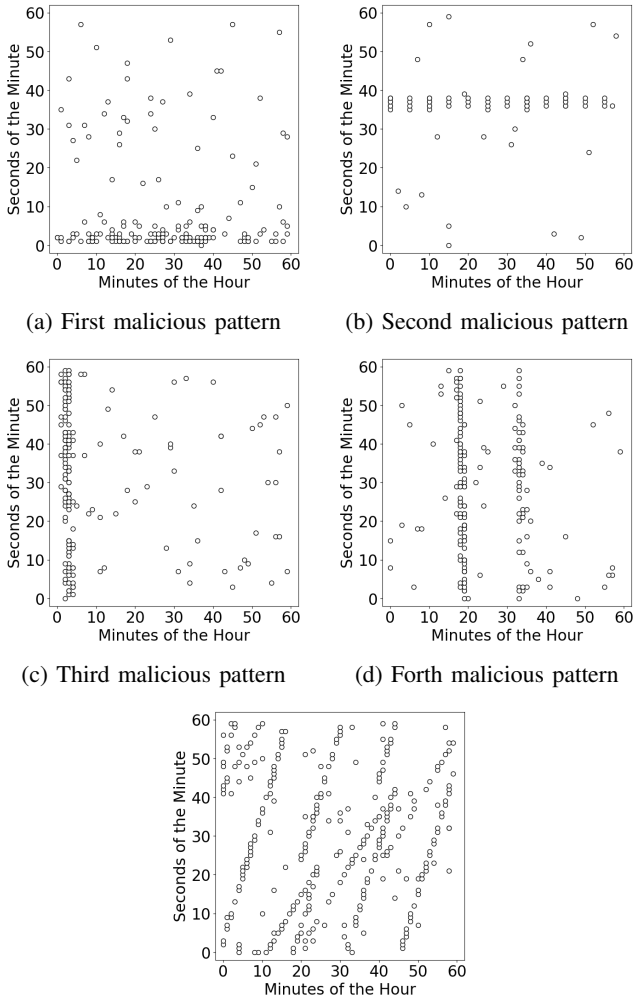
Fig. 2: Project framework

Human, benign OSN applications and malicious OSN applications in some NSP can send requests to OSN severs. When OSN severs receive their requests, OSN severs will reply responses to the people/applications who sent requests in previous NSP. 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/human within its network and the OSN severs. 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 only can be decided 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 is that malicious OSN API-based applications post/comment spam contents (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 behave patterns probably be different from benign accounts. There are 5 known malicious patterns for malicious API application controlled spam/bot accounts on Twitter [21]. Those accounts post/retweet/like behavior time points on Twitter follow 5 malicious patterns in figure 3 . Therefore, corresponding malicious API-based applications post/retweet/like time points



(a) First malicious pattern

(b) Second malicious pattern

(c) Third malicious pattern

(d) Forth malicious pattern

(e) Fifth malicious pattern

Fig. 3: Malicious API-based application controlled OSN accounts behave time points follow 5 malicious patterns

should also follow those 5 known malicious patterns. For those 5 malicious patterns in figure 3, x axis indicates minutes of the hour for an account post/comment/like/retweet behavior time point, while y axis indicates the seconds of the minute for this account post/comment/like/retweet behavior time point.

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 two most common malicious API-based behaviors in this project:

- API-based post/comment spam contents for a single account.
- API-based change accounts to post/comment spam contents.

In this project, if any of those three API-based behaviors post spam posts/comments following any of 5 known malicious API usage patterns, we define it as malicious API-based application behavior.

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

When OSN released APIs are used with 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. For example, "Flow" is a Microsoft application which integrates Facebook API, Twitter API and Instagram API. It can help people post at 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 being post and comment patterns. It has been found that human post/comment/like time points on Twitter can be modeled as different Poisson Processes [21]. This benign pattern can be converted to the pattern in figure 4 which is drawn in a similar way with 5 malicious patterns based on human post/comment/like/retweet time points. The benign API-based application behave time point patterns should also follow this benign human post/comment pattern on OSNs.

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

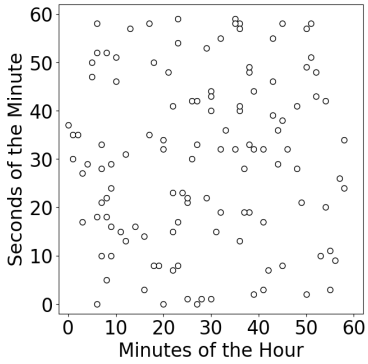


Fig. 4: Benign API-based application controlled OSN accounts behave time points follow benign pattern

3) Human OSN behaviors: In most cases, users visit social networks, and perform some normal operations using browsers instead of using APIs. Human being usually access OSNs on browsers to do various operations, such as login, post, comment, browser 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 those flow data generated by human operations as benign flows.

To sum up, we define three behaviors: malicious API-based application behaviors, benign API-based application behaviors and human OSN behaviors in Section IV. 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. Human can do 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 aims to detect flows of malicious API-based applications by detecting flows with malicious features.

V. Data Generation

Since there is no existing, available data set providing flows for malicious API-based application behaviors nor other benign OSN behaviors, we need to generate those data by ourselves. As it's described in problem formulation Section 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 behaviors follows 5 malicious API usage timing patterns to post/-comment malicious contents separately, and change related parameters to simulate various malicious API usage timing patterns. To generate flows for benign API-based application behaviors, we have different API-based OSN behaviors follow benign API usage timing patterns to post with benign information, and change related parameters to simulate various benign timing patterns. In addition, we simulate all kinds of human operations on OSNs using browsers, and collect those behaviors generated flows as benign human OSN behavior flows.

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

A. Data generation platform

As described in research method section, 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, 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 below 3 defaulted flow collection parameters.

- 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 long 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.
- 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 a same flow. If the interval of two consecutive packets is larger than Expint time, they will be classified into different flows.
- 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 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 on this testbed.

B. Synthesize malicious API-based application behaviors, benign API-based behaviors and human OSN behaviors

1) Synthesize human OSN being behaviors: Since most OSN flows are generated by normal human operations on OSNs, it is very important to synthesize normal human being behaviors accurately, then can we collect human OSN behavior flows accurately as ground truth.

In order to simulate normal human being behaviors comprehensively, we write scripts to simulate all kinds of human user behaviors on WordPress, e.g. login, browse, post, comment by browser. Table I shows all human basis operations on WordPress. Human being operations on OSNs are actually driven by a series of click events on the browser. To simulate human being 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[23].

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

TABLE I: Aggregated flows data size for each malicious/benign patterns

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 then comment, no login and comment as a stranger
Post	Login then post

scripts to simulate 27 streamlines of human behavior chains based on event occurrence sequences, which almost cover all possible user operation streamlines on WordPress in the real word expect 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 theirs implementations by changing theirs related parameters, so our simulation nearly cover all possible human behaviors on this small social networks accurately and comprehensively.

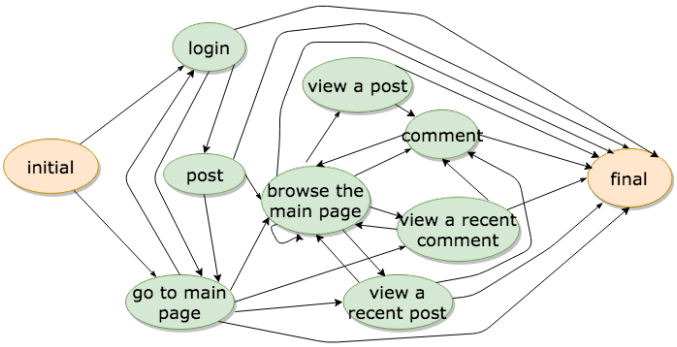
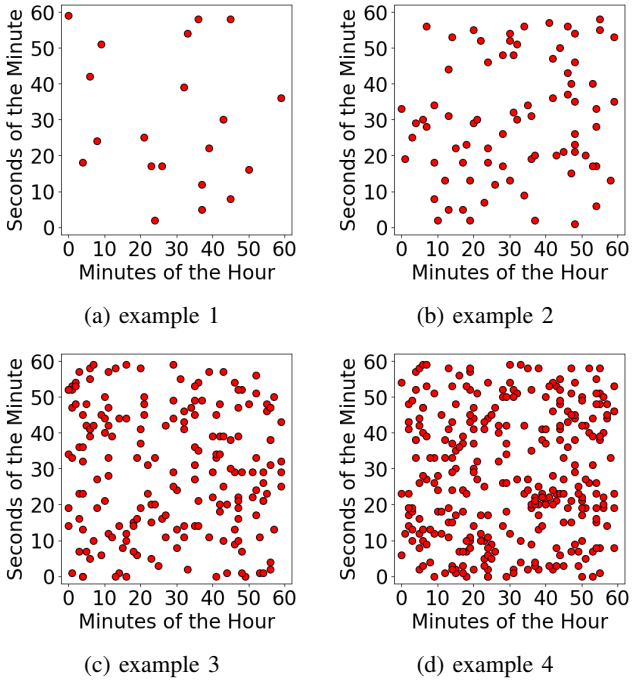


Fig. 5: Streamline of how different events switch to each other on WordPress

2) Synthesize 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 softwares. The third party software are supposed to use API provided OSN services serve human beings automatically when people need. Therefore, the benign API third party applications behavior 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 [21], so benign API-based benign third party application behavior time points should follow Poisson Process. Except 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 general people. For those benign bot applications, they will post contents on a social network whenever there is a news or warning, and those news and warnings occurrence pattern also can be modeled as Poisson Process, so benign API-based bot applications behave pattern can also be modeled as Poisson Process. To sum up, all benign API-based applications behaviors should follow Poisson Distribution.



(a) example 1 (b) example 2

(c) example 3 (d) example 4

Fig. 6: Simulate 4 benign API-based applications by changing Poisson parameters

If API-based application behaviors 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, and their behavior time points follow various benign Poisson processes by changing different parameters. Figure 6 shows 4 examples of our simulated benign API-based application behavior timing patterns with four different Poisson parameters.

3) Synthesize malicious API-based application behaviors: Malicious OSN API-based applications behave with malicious purposes to post malicious data, and those application behavior patterns probably be different from benign accounts. As we discussed in IV Section, there are 5 known malicious API-based application behave timing patterns, and we also have

defined two categories of most common API-based behaviors as API-based post/comment spam contents for a single account, API-based change accounts to post/comment spam contents.

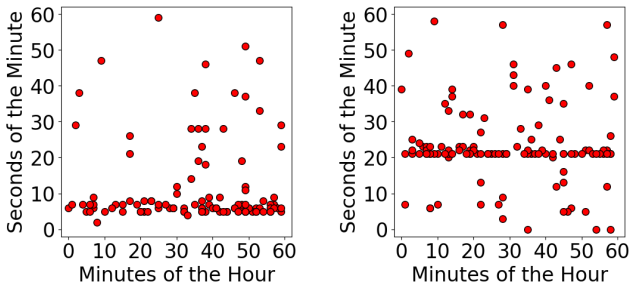
We download a twitter dataset which provides malicious contents posted or commented by malicious spam accounts. To simulate various malicious API-based application behaviors, for each API-based OSN behaviors, we have its behavior time points decided by five known malicious timing patterns, and it will post or comment malicious contents at its every behavior time points. We model each malicious timing pattern as combinations of different probability distributions, and simulate each malicious timing patterns 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 same λ value. Therefore, we can get 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. Parameter λ can decide the shape for all exponential distributions.

To simulate various implementations of first malicious API usage timing patterns, we change parameters λ, $P0$, $M0$, 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 same δ. The third malicious API usage timing pattern probability density is a combination of a uniform distribution and a group poisson distributions with same δ. The forth 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 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 five malicious API usage patterns are shown in formula 1, 2, 3, 4 and 5 respectively.

As we described above, there are two categories of most common API-based malicious behaviors, and each behavior post/comment malicious contents, and theirs behavior time points are decided by any of 5 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 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 application behaviors post/comment various malicious contents with 5 known malicious patterns, and our simulation result is able to cover nearly all possible 5 known malicious API-based application behaviors.



(a) example 1 (b) example 2

(c) example 3 (d) example 4

Fig. 7: 4 simulation results for the first malicious API-based application

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

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

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

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

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

C. Synthesize flows for malicious API-based application, benign API-based application and human OSN behaviors

In above V-B Section, 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, more details about how to synthesize flows for three behaviors will be introduced.

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 how users behaviors both in user active time and user waiting time. During user active time, users visit OSNs actively, and do various of 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, their waiting time is much longer than their active time. Therefore, ground truth generation speed is very slow because of a lot of time wasted on simulating user waiting time.

However, during user waiting time, there is 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 / human behavior on OSN can generate several flows. Even if API / human behavior time points are very close, a same API / human behavior occurred at different time points can generate different flows. Not only flow numbers generated by the same behavior could be different, but also 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/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) Synthesize flows for malicious API-based applications and benign API-based applications: Our methodology of how to synthesize flows for automated malicious and benign API usage behaviors in a long time period follows below logic.

1) To begin with, collect flows generated by a single post/-comment API operation. Benign applications will post with benign contents, while malicious applications will post/comment with malicious contents. Save those flows into a file, then each file includes all flows generated by this single API operation.
2) In a similar way, repeat multiple single API post/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 time and end time while keep all other attributes same. Time intervals are generated based on which malicious timing pattern and benign timing pattern this benign/malicious API-based application follows.

Based on above method, the generated large flow file is the synthesized flows for malicious/benign API behaviors in a long time period, and we can generate flows for each malicious/benign API-based application as long as needed.

2) Synthesize flows for human OSN behaviors: The method of synthesizing human OSN flows is a little different from synthesizing flows for malicious/benign API applications. For API-based applications, their behaviors mainly consist of different API-based post, comment behaviors. However, human operations on social networks are much different, so we collect user follows based on single user session instead of a single operation on social networks. A single user session includes all activities from a user open a OSN webpage to close it, and the user may do various operations on OSNs at each user Section discussed in V-B1. The user session time duration distribution and click event pattern follow the pattern summarized in a real Chinese social network RenRen[23].

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 time and end time, Our methodology of how to synthesize flows for human behaviors in a long time period follow below logic.

1) Firstly, collect flows generated by a single user session, then save those flows into a file, then 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' start time and end time while keep other attributes same. Human being uses social network intervals follow Poisson distribution, so we use Poisson intervals here.

In this way, the generated large flow file is the synthesized flows for human behaviors in a long time period, and we can generate flows for human OSN behaviors as long as needed.

3) How we control flow attributes when generating synthetic flows: For each flow, it includes several attributes, such as flow start time, flow end time, flow duration, source IP, source port, destination IP, destination port, TCP flag, packets number in each flow, packets number in each flow. In this section, we will discuss what attributes of flow are controlled 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/comment behaviors, the second is to decide what content they post/comment each time. For malicious OSN applications, their post/comment time points are decided by 5 malicious timing pattern 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/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/benign OSN application behaviors. Now we will answer the question how we control flow attributes for benign/malicious OSN application behaviors. To begin with, for malicious OSN applications, 5 malicious timing patterns decide flows' start time, and the behavior of "post/comment spam contents for 1 or multiple accounts" decides flows' other features. After that, for benign OSN applications, 1 benign timing pattern decides flows' start time, and the behavior of "post/comment benign contents" decides 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 open a OSN webpage to close 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 a real Chinese social network RenRen[23]. Since a human user may visit social network several times a day, a human user may generate several user sessions. The intervals between different user sessions follow 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 sum up, in Section 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 time benign/malicious API-based application behaviors and human OSN behaviors. We also analyze how we control flow level features for different synthesized flows.

VI. Deep Learning based Model

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 session, we will introduce how to use our generated ground truth to train a Convolutional Neural Network (CNN) model. This section includes some basis works, such as 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 of machine learning tasks in recent years. It is particular 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 aggregated flows in our project contains different number of flows. Therefore, we normalize each aggregated flows by transforming each aggregated flow into an image, while each image still reserves the main and useful information of each aggregated flow group, then use normalized images as the input to train the CNN model.

B. Ground truth labeling

There are three OSN behaviors defined in this paper: 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 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, convert each aggregated flow into an image, which can act

TABLE II: Aggregated flows data size for each behaviors

Data type	Data size
Automated malicious API usage behaviors	15,000
Automated benign API usage behaviors	15,000
Human behaviors	10,000

TABLE III: Aggregated flows data size for applications with each malicious/benign application pattern

Data type	Data size
Malicious bad1	3,000
Malicious bad2	3,000
Malicious bad3	3,000
Malicious bad4	3,000
Malicious bad5	3,000
Benign good0	15,000
Human good1	10,000

as the normalized input of CNN model. We generate 40,000 aggregated flows for both benign behaviors and malicious API application behaviors, and the date size of categorized aggregated flows for each behavior is shown in table II.

As we talked about in data generation section, there are five malicious API-based applications. They post/comment 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, human behaviors as good0. Table III has shown generated aggregated flows size for applications each malicious timing patterns and benign timing patterns. The data set is split into training set and test set. The training set has 80% of all labeled aggregated data, while the test set includes 20% 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 trained model.

D. Flow data preprocessing

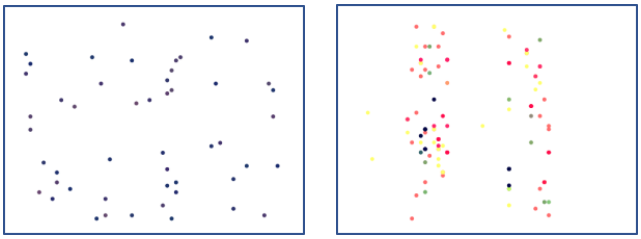
For flows generated by each application/human behaviors, we will preprocess flows first, then train deep learning model using preprocessed flows instead raw flows. The flow preprocessing process includes several steps: flow extraction, aggregation, and normalized aggregated flows into images, and normalized images will be act as the input of 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 softwares, so it can generate and collect flow data based on current incoming and outgoing packets on this client. However, when we simulate our those three kinds of behaviors 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 Wordpress sever, but also includes flows generated by other applications on the client. Therefore, we need to extract those flow data which generated by communication traffic between this client and Wordpress server. To collect those flows generated by our synthesized behaviors, we extract those flows whose source IPs or destination IPs belong to the Wordpress server. Our IP address matching method can also extract flows generated by any other OSNs, such as Twitter and Facebook.

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

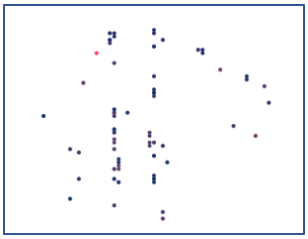
Since aggregated flows have different number of flows, and CNN model needs uniform input size, we normalize the input of each aggregated flows by converting it to an image which carries main and useful information for the aggregated flows. Each aggregated flow group is converted into a scatter image, 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 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, packet number attribute decides the point's G value, while TCP flag decides the point's B value. In this way, the four main useful attributes of a single flow data are carried by 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. 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 in figure 8.



(a) Converted image for benign API-based applications

(b) Converted image for human behaviors



(c) Converted image for malicious API-based applications

Fig. 8: Converted images for human behaviors, malicious API-based application and benign API-based application

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 previous layer is the input of next layer as in figure 9. The first layer is the input of the model, and the last layer is the prediction result of the model. 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 output layer. If it's predicted with 0, it indicates that the flows are generated by benign benign API-based application behaviors or human being behaviors on OSNs. If it's predicted with 1, it indicates that the the flows are generated by malicious API-based application behaviors.

To sum up, in this section, we describe how to label data, preprocess flows into images to train CNN model, and trained CNN model structure.

VII. Evaluation

In this section, we evaluate the performance of our trained CNN model. Four metrics accuracy, recall, precision and F1-measure are used to evaluate the detection performance of trained CNN model. 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/comment 10 times, 20 times, 30 times, 40 times, 50 times. Afterwards, to get a better understanding of how can our CNN model detect each malicious API-based application behaviors effectively, we evaluate CNN's detection performance when each malicious API-based application posts/comments 10 times, 20 times, 30 times, 40 times, 50 times. At last, we evaluate the performance of trained CNN model by detecting flows generated by 3 real world API-based malicious applications and 3 API-based benign applications.

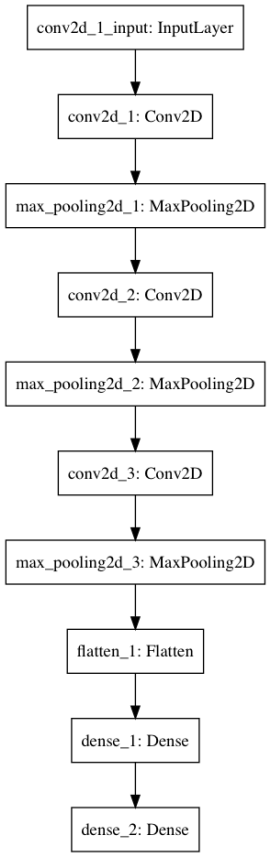


Fig. 9: CNN model structure

A. Test set size and evaluation metrics

We reserve 8,000 (20% of generated ground truth data) aggregated flows as 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, and their corresponding data size are shown in table IV.

TABLE IV: Test set data size for each behaviors

Test set data type	Data size
Malicious API-based applications	3,000
Benign API-based applications	3,000
Human OSN behaviors	2,000

We adopt four most common used metrics accuracy, recall, precision and f1-measure to evaluate the performance of our trained CNN model. Accuracy is the proportion of all predictions that are correct. Recall is a measurement of how many actual positive observations are predicted correctly. Precision measures how many positive predictions are actual positive observations. F1-Measure is the harmonic 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. Those 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 trained CNN model by predicting the whole test set. As it's shown in Fig 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.

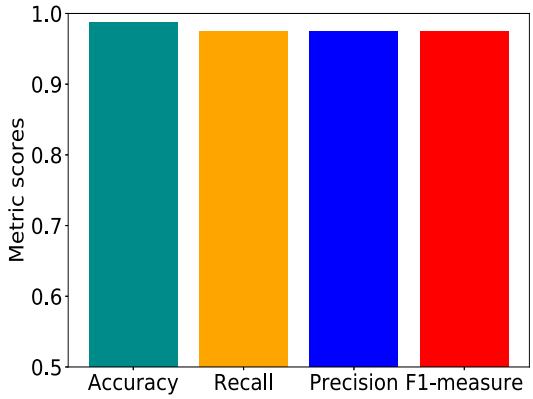


Fig. 10: Detection performance for predicting all test set

C. Detection performance for malicious API-based applications with different post/comment frequencies

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

In figure 11, x axis indicates malicious API applications post/comment total times in an aggregated flows, 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. When malicious patterns post/comment more and more frequently, our model's detection accuracy is increasing, and it will reach up to nearly 99.3% when malicious APIs post 50 times.

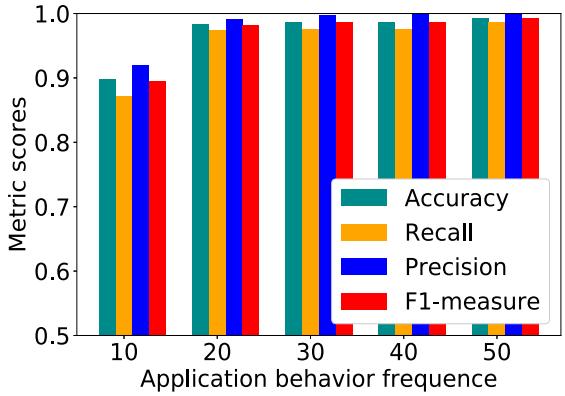
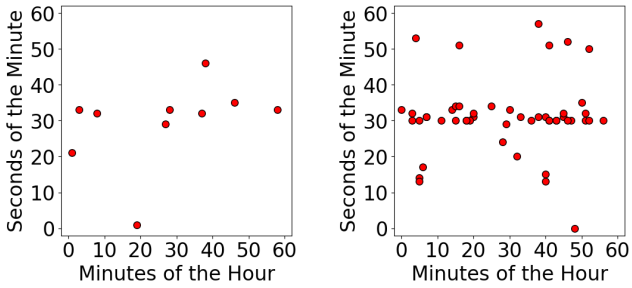


Fig. 11: Detection performance for malicious API usage with different post/comment times



(a) First malicious API application posts 10 times

(b) First malicious API application posts 50 times

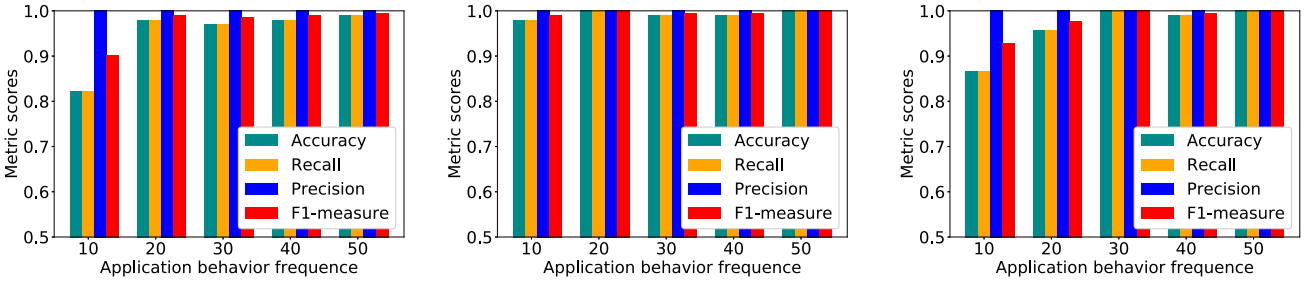
Fig. 12: Comparison for first malicious API application posting 10 times and 50 times

When malicious API-based applications behave more and more frequently, their malicious patterns are more and more obvious, then their flows are more and more easily to be 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 the malicious pattern is not that obvious. In figure 12b, the malicious pattern is much more obvious when it posts 50 times than in 10 times. Therefore, we can find when malicious API-based applications post times is increasing, their malicious patterns will be more and more obvious. Our method can detect flows from frequent behaved API-based applications more effectively.

D. Detection performance for each malicious API-based application with different post/comment frequencies

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

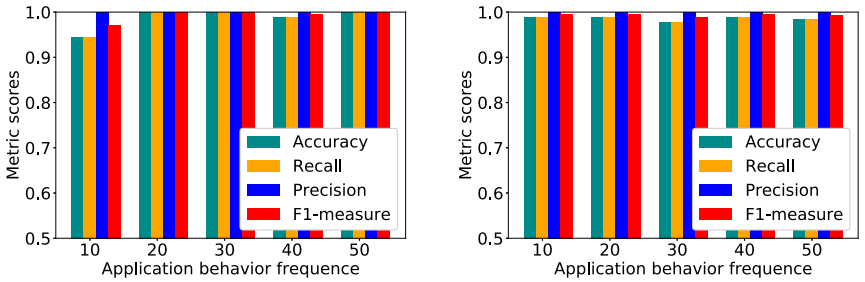
Figure 13 shows the performance of our trained model in detecting flows of each malicious API applications when their posting times change from 10 times to 50 times. Based on



(a) Detection results for first malicious API-based application

(b) Detection results for second malicious API-based application

(c) Detection results for third malicious API-based application



(d) Detection results for forth malicious API-based application

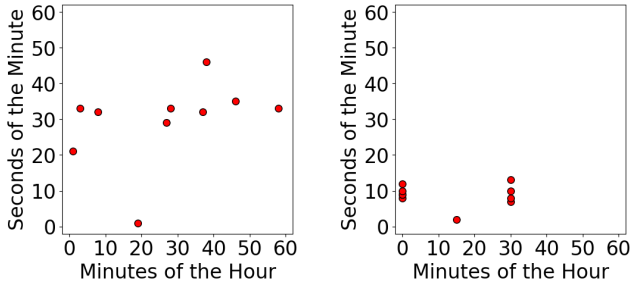
(e) Detection results for fifth malicious API-based application

Fig. 13: Detection results for each malicious API-based application posting/commenting from 10 times to 50 times

our observation, we found that the detection result shows two patterns. For malicious API-based application bad1 and bad3, when their post/comment times is less frequently, our model's detection accuracy is very low. When their post/comment times is becoming more frequently, our model's detection accuracy is increasing until very high. However, for malicious API-based application bad2, bad4 and bad5, our model's detection accuracy is always very high.

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

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 a measurement of how many positive predictions are actual positive observations. We represent malicious API behaviors as positive, which is described in section VI. This high precision result indicates that our model's predicted malicious flows are very likely to be actual malicious flows. Our model may predict malicious flows as benign flows. When the behavior pattern of some malicious API applications is not obvious, our model can mistakenly predict those malicious flows as benign.



(a) Bad1 application posts/comments 10 times

(b) Bad2 application posts/comments 50 times

Fig. 14: Comparison for bad1 application and bad2 application for posting 10 times

E. Detection performance for detecting real world API-based benign applications and malicious applications

Since we have evaluated the performance of trained model based on five synthesized malicious applications, in this section, to demonstrate that our synthetic flows are very close the real world generated malicious and benign flows, we use our trained CNN model to detect flows from 3 real world malicious OSN API-based applications and 3 real world benign OSN API-based applications.

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

We collect and aggregated corresponding flows by running

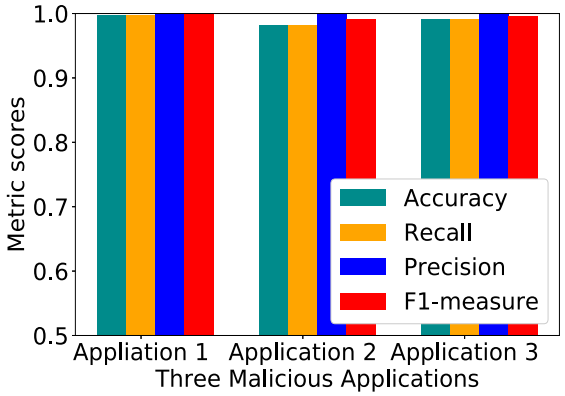


Fig. 15: Detection results for flows generated by 3 real world malicious OSN API-based applications

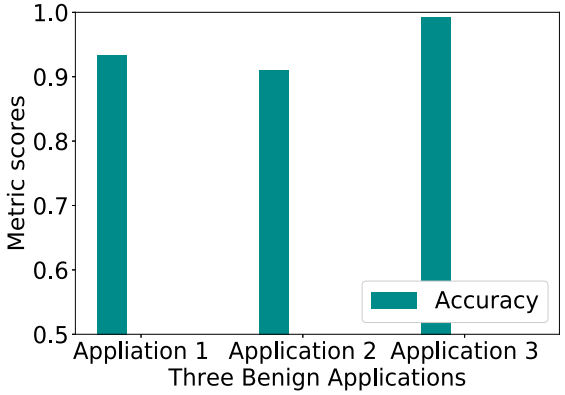


Fig. 16: Detection results for flows generated by 3 real world benign OSN API-based applications

the those applications on our platform, then use our trained model to detect aggregated flows generated by 3 malicious applications and 3 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 can not be calculated because of no true positive TP and false negative FN, while all samples are correctly predicted as TN, and mistakenly predicted as FP. The figure can show that our model can detect flows from real world benign and malicious API-based applications well.

To sum up, our trained CNN model might be not able to identify flows for some malicious API-based applications very accurately when they post/comment 10 times. When their post/comment times increase to 20 times, 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.

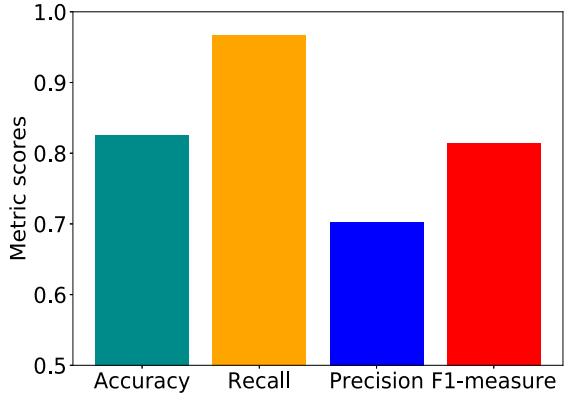


Fig. 17: Detection result for predicting all test set for model with only timing feature

VIII. Discussion

In this section, we are going to discuss some concerns and the future work for this work. 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 we are not necessary to train a deep learning model based on other features. The second is that people may question that our project can only detect flows from malicious OSN programs that demonstrate a similar timing patterns with our used 5 malicious timing patterns. For the future work, we are going to discuss how to use our work in reality to help 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 malicious purpose of spreading spam information, so their behavior pattern are probably different with benign applications and human behaviors. It's known malicious OSN applications behavior timing patterns may be different with benign applications and human behaviors based on previous paper, and some people may concern that a single timing pattern feature is enough to detect flows generated by malicious OSN applications. In this project, we train a deep learning model based on four features: flow start pattern, 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 timing feature, and found that the detection result for the whole test set is in figure 17.

As we can see, the detection accuracy of 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 malicious 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 achieve accuracy as high as 98.7% and precision as high as 97.5%.

TABLE V: Detection accuracy result for different flows 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%

To get a better understanding of why timing feature based model gets overall low accuracy and low precision, we investigate four questions, and train 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 all benign application flows, malicious application, and human flows.

In the table V, timing feature based model 1 is trained by malicious application flows and benign application flows. We can 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 application and human behaviors, we can find that our trained model 2 can label malicious application flows with accuracy 99.6%, while label flows from human behavior only with accuracy 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, its 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 can find that it can distinguish benign and malicious application flows with very good performance, but human flows with very low accuracy. Based on model 1, we know timing feature is enough to distinguish benign application flows or malicious application flows. From model 3, we know we know timing feature is enough to distinguish benign application flows or human flows. In model 2, human flows can be mislabeled as malicious flows only by detecting timing feature. Model 4 gets a low accuracy in detecting human flows, and this is also caused by some human flows 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 same with model 2. In model 2, the model has already has already had a very hard time to distinguish human flows from malicious application flows only based on timing feature. It can be understandable for model 4 having a bad performance to distinguish bad flows from benign and malicious application flows too.

We can conclude based on table V: timing feature can?t distinguish between human and malicious applications, but help distinguish between malicious and benign apps and benign apps and humans. Timing feature is an important feature to detect malicious flows, but a single timing feature is not enough to detect flows from malicious application, because it can mislabel human flows as malicious, and this will get a high false positive rate.

TABLE VI: 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%

We have already found that timing based model 4 with a low detection accuracy and precision is mainly caused by labeling some human flow 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 in particular helpful? Does a combination of all useful features can further improve the detection result?

In our project, the malicious application flows detection model is trained based on timing pattern and another three 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 by the timing feature and other

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.

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

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

TABLE VII: Detection accuracy result for different flows in each four-feature based model

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

Based on above analysis, for model 2 trained under the presence of only malicious and benign application flows, it can't achieve good performance when it?s only trained based on a single timing pattern. If we train it with timing and all other 3 features in model 8, the detection performance is very good.

We also train a model with timing and other 3 features to distinguish flows from malicious or benign applications in model 9 in table VII, and model 9?s detection performance in table VII is also very good. Model 10 is trained by timing and other three features to distinguish human flows and benign application flows, and the detection performance.

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

At last, model 11 is trained based on timing and another 3 features to distinguish malicious application flows, benign application flows and human flows, the detection performance for all flows is also very good. Therefore, a timing feature is not enough to distinguish all three flows, and timing and 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 used 5 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 VIII-A section, 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, other three features are also helpful. In a more accurate way, this project is to identify applications that exhibit both 5 malicious timing patterns and other flow-level malicious features, instead of identifying applications that only exhibit 5 malicious timing patterns.

In this project, we have 5 malicious timing patterns decide when malicious OSN applications post/comment with spam contents, so our trained model is indeed supposed to detect flows from malicious applications showing a similar timing patterns with our used 5 malicious timing patterns.

In order to create an expressive set of malicious timing patterns, we extend the findings of paper [21]. 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, so it is no surprise that our CNN successfully detected the flows generated by these spam programs as malicious.

To sum up, we vary all related parameters to simulate all possible malicious timing pattern instances, and our simulated

instances can cover some timing patterns in the real-world malicious programs. If the real world malicious program behavior timing pattern is covered in our dataset, our trained model can detect this malicious application generated flows.

C. Future work

Our project is proposed to detect flows generated by malicious OSN API-based applications for NSPs. In this session, we will discuss how to use our work in reality to help NSPs to detect flows generated by malicious OSN applications.

To make use of our work in the real world, NSPs should deployed a 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 cost that much efforts, because flows data only aggregated packet head information, and the flow data size is much small than all packets size. If a flow caught by the NSP has its source IP address or destination IP address belong to a OSN, then this flow is generated by a connection between a machine inside the NSP and a OSN. There are real time online router tables that can provide IP blocks for a OSN, so NSP can decide whether a flow's source or destination address belongs to a OSN by IP prefix matching. In this way, NSPs can get all traffic flows generated between machines inside its network and a 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 malicious, it indicates that the machine with this IP is running malicious OSN applications, then the NSP can decide to block the bad traffic, block this compromised IP, or just don't do anything.

To use our proposed method to train a deep learning model for detecting malicious flows, NSPs need to get 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 ground truth for this OSN, and they cannot train this model automatically. This could be a disadvantage of our work.

IX. Conclusion

Most social network providers release some APIs for third party developers to integrate OSN services to their own softwares. However, those APIs are misused widely by malicious applications, such as bot applications, crawler applications, even third-party applications to spread spam information or secretly collect private user data. It is important and necessary to monitor and detect flows generated by those malicious API-based applications for NSPs. This paper is the first research to enable the detection of flows for OSN malicious API-based applications for NSPs at the network flow level. This paper aims to detect flows generated by 5 known malicious API-based application behaviors. To achieve this goal, we collect flows various human behaviors, various benign API-based application behaviors, and 5 malicious API-based application behaviors, then aggregate and label network flows for each behaviors, and train a deep learning model based on labeled ground truth. Our evaluation result shows that the trained deep learning model is able to detect flows generated by malicious API-based applications with 97.6% accuracy, and 1.6% false positive.

References

[1] N. C. Matthew Rosenberg and C. Cadwalladr, "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

[2] X. Zheng, Z. Zeng, Z. Chen, Y. Yu, and C. Rong, "Detecting spammers on social networks," *Neurocomputing*, vol. 159, pp. 27–34, 2015.

[3] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on twitter," in *Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)*, vol. 6, no. 2010, 2010, p. 12.

[4] A. Almaatouq, E. Shmueli, M. Nouh, A. Alabdulkareem, V. K. Singh, M. Alsaleh, A. Alarifi, A. Alfaris *et al.*, "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*, vol. 15, no. 5, pp. 475–491, 2016.

[5] M. Fazil and M. Abulaish, "A hybrid approach for detecting automated spammers in twitter," *IEEE Transactions on Information Forensics and Security*, 2018.

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

[7] S. H. Ahmadinejad and P. W. Fong, "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*. ACM, 2013, pp. 161–166.

[8] R. Singh, H. Kumar, and R. Singla, "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*, 2012, pp. 163–166.

[9] J. François, S. Wang, T. Engel *et al.*, "Bottrack: tracking botnets using netflow and pagerank," in *International Conference on Research in Networking*. Springer, 2011, pp. 1–14.

[10] A. Kind, M. P. Stoecklin, and X. Dimitropoulos, "Histogram-based traffic anomaly detection," *IEEE Transactions on Network and Service Management*, vol. 6, no. 2, pp. 110–121, 2009.

[11] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. ACM, 2001, pp. 69–73.

[12] W. Zhenqi and W. Xinyu, "Netflow based intrusion detection system," in *MultiMedia and Information Technology, 2008. MMIT'08. International Conference on*. IEEE, 2008, pp. 825–828.

[13] R. Hofstede, V. Bartos, A. Sperotto, and A. Pras, "Towards real-time intrusion detection for netflow and ipfix," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, 2013, pp. 227–234.

[14] D. van der Steeg, R. Hofstede, A. Sperotto, and A. Pras, "Real-time ddos attack detection for cisco ios using netflow," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 972–977.

[15] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing user navigation and interactions in online social networks," *Information Sciences*, vol. 195, pp. 1–24, 2012.

[16] W. Wongyai and L. Charoenwatana, "Examining the network traffic of facebook homepage retrieval: An end user perspective," in *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*. IEEE, 2012, pp. 77–81.

[17] Z. Móczár and S. Molnár, "Comparative traffic analysis study of popular applications," in *Meeting of the European Network of Universities and Companies in Information and Communication Engineering*. Springer, 2011, pp. 124–133.

[18] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding online social network usage from a network perspective," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. ACM, 2009, pp. 35–48.

[19] A. H. Wang, "Don't follow me: Spam detection in twitter," in *Security and cryptography (SECRYPT), proceedings of the 2010 international conference on*. IEEE, 2010, pp. 1–10.

[20] A. Saroop and A. Karnik, "Crawlers for social networks & structural analysis of twitter," in *Internet Multimedia Systems Architecture and Application (IMSAA), 2011 IEEE 5th International Conference on*. IEEE, 2011, pp. 1–8.

[21] C. M. Zhang and V. Paxson, "Detecting and analyzing automated activity on twitter," in *International Conference on Passive and Active Network Measurement*. Springer, 2011, pp. 102–111.

[22] J. Herrera-Joancomartí and C. Pérez-Solà, "Online social honeynets: trapping web crawlers in osn," in *International Conference on Modeling Decisions for Artificial Intelligence*. Springer, 2011, pp. 1–16.

[23] G. Wang, X. Zhang, S. Tang, C. Wilson, H. Zheng, and B. Y. Zhao, "Clickstream user behavior models," *ACM Transactions on the Web (TWEB)*, vol. 11, no. 4, p. 21, 2017.

Detecting Malicious Usage of Online Social Network APIs from Network Flows

Dan Li
Computer and Information Science
University of Oregon
Eugene, US
dli@cs.uoregon.edu

Jun Li
Computer and Information Science
University of Oregon
Eugene, US
lijun@cs.uoregon.edu

Lei Jiao
Computer and Information Science
University of Oregon
Eugene, US
jiao@cs.uoregon.edu

***Abstract*—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 paper, 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 NetFlows 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% NetFlows from the malicious OSN applications, with only 1.6% false positives.

***Index Terms*—online social network (OSN); NetFlow; network flow; malicious OSN application; OSN application; OSN API**

I. Introduction

Online social networks (OSN) have become extremely popular with an ever-growing user base. At the time of writing this paper, 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 [1].

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 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 [2]–[5], limit large-scale crawling activities [6], detect malicious third-party OSN applications [7], 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 words, 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 paper is organized as follows. We first cover related work in Section II, then describe the problem and solution in Section III, followed by Section IV that illustrates the data (the traffic flows) used in this study. We evaluate the performance of our solution in Section V and conclude our work in Section VI.

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 ([8], [9] and [10]). The work in [8] crawled a large amount of sensitive OSN data by using OSN APIs, and research in [9] 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 [10] 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 [2], [3], [4] and [5]. 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 [11] and [6] proposed countermeasures to prevent attackers from crawling sensitive OSN user data. Research in [11] proposes an "Online Social Honeynet" 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 [6] 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 [12] uses campus traffic flows to detect anomaly broadcast traffic, while [13] extends the popular PageRank algorithm to detect botnet traffic. The work [14] and [15] both detect network traffic flow anomalies by analyzing flow-level anomaly features. And research in [16], [17], and [18] 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 [19]–[22] 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.

III. Methodology

In this section, we describe our proposed methodology which enables an NSP to detect flows generated by malicious OSN applications from within its network.

We first describe the settings of this problem. As shown in Fig. 1, a network service provider (NSP) can have human users, benign OSN applications, and malicious OSN applications send requests to OSN severs. When an OSN sever receive such requests, it will then respond the requests. Both requests from an NSP to an OSN server and responses from an OSN server to the NSP will go through the NSP's border router. In this situation, the border router can collect flows generated by the applications and human users from within the NSP network, and we thus assume that our malicious OSN flow detection method is deployed at the border router of the NSP. Once it detects some malicious flows at the border router, it then can send alerts to the NSP.

We hypothesize that traffic flows of human users of OSN, benign API-based OSN applications, and malicious API-based OSN applications have different flow-level features. We now describe how we extract, aggregate, and visualize OSN flow data and train a deep learning model to detect flows from malicious OSN applications.

A. Flow Extraction

When a machine inside an NSP visits an OSN, typically this machine also runs other applications at the same time. Therefore, the border router of the NSP not only collects network flows from this machine visiting an OSN, but also those flows from other applications on this machine (e.g., communications with other web servers). We thus need to

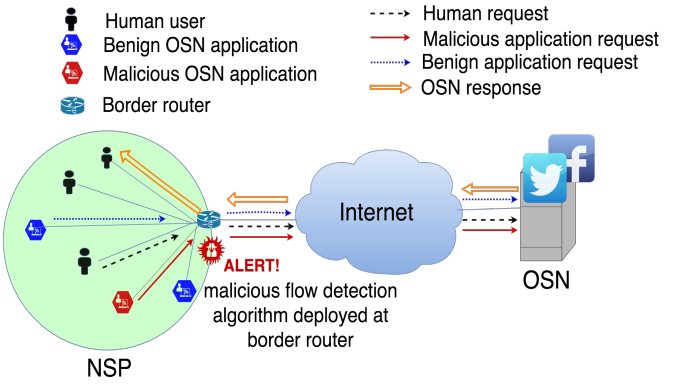


Fig. 1: Setting of a network service provider (NSP) in detecting malicious OSN network flows

extract those flows generated by traffic between this machine and OSN servers, in order to further decide which those flows is malicious.

The procedure is as follows. We first extract all flows generated by a single machine in an NSP by extracting all flows whose source or destination IP address belongs to this machine. From these flows, we then further extract all flows between this machine and specific OSN providers. We obtain the real-time IP address blocks of an OSN provider by sending requests to BGPstream [23] to check real-time router tables. If a flow's source IP or destination IP belongs to the IP blocks of the OSN provider, this flow is an OSN flow generated between this machine and this OSN provider. As a result, we can extract all flows generated between a machine in an NSP and an OSN provider.

B. Flow Aggregation

After extracting OSN traffic flows for a machine, we aggregate flow data. A single simple operation on an OSN can generate several traffic flows, so a single traffic flow could hardly carry any information about how a user operates on an OSN. Therefore, we aggregate traffic flows that occur in a relative long pre-specified time window, such that the aggregated flows can indicate how user behaved in the time window. If an API-based application is running in this pre-specified time window, we will detect those aggregated traffic flows from this application as malicious.

C. Flow Visualization and Model Training

Deep learning is a popular tool to automatically learn hierarchical features from complex, high-dimension data and then conduct a classification task effectively. As aggregated flows in our research are high-dimension data, deep learning is a suitable tool to classify malicious API flows from benign flows and human user flows. We visualize each aggregated flow and transform it into an image, where each image preserves the main and useful information of each aggregated flow group, and then use images as the input to train a Convolutional Neural Network (CNN) model.

Each aggregated flows is converted into a scatter image, where each point in the image carries information about four main useful attributes from a flow by analyzing real-world campus flow traffic: flow start time, flow duration, packet number, and TCP flag. Flow start time decides the location (x, y) of a point in this image: x-axis is the time of minutes when the flow is generated, while y-axis is the time of seconds when the flow is generated. The flow duration decides the point's R value, and the packet number determines the point's G value, and the TCP flag determines the point's B value. In the end, these four main attributes of a flow are converted to a point's location and (R,G,B) color value in the image, and the image eventually carries all the points converted from all flows in each aggregated flow group, so the image can carry all useful information from all flows in an aggregated flow group. Fig. 2 shows three converted images of aggregated flows from malicious API-based applications, benign API-based applications, and human operations separately.

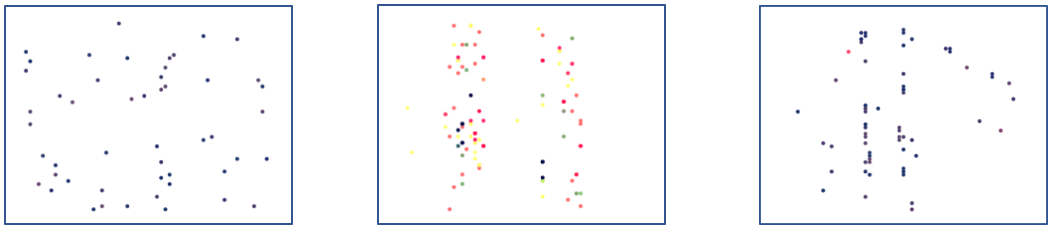
For each transformed image, we import its information into a 128*128*3 dimension matrix as input to train the CNN model. By changing all related parameters and layers, we can obtain a CNN model that can achieve best flow classification results. For each input data, the trained CNN model can predict it with label zero or label one at output layer. If label zero, it indicates that flows are generated by benign applications or human users on OSNs; otherwise, it is label one and flows are generated by malicious applications.

IV. Data

Since there is no available dataset of network flow data generated by malicious API-based OSN applications and other benign OSN flows, we emulate and collect flows for human user operations, malicious OSN applications, and benign OSN application on a small social network testbed WordPress, and implement our proposed solution based on flows collected from this testbed. Our proposed methodology not only works on the deployed testbed, but can also be applied to detect flows generated by malicious applications on any other OSNs, such as Twitter, Facebook, Instagram. Below we first introduce our data generation platform, then define, emulate, and collect flows for the above three types of flows, and lastly, describe how we optimize flow generation.

A. OSN Testbed

We deploy a small social network WordPress system as our testbed 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 machine configured with 8GB Memory, 128GB Disk, and Ubuntu 16.04 Operating System. The client is installed with a traffic flow generation software *softflowd* and a flow collection software *nfdump* to generate



(a) Converted image for benign application flows

(b) Converted image for human user flows

(c) Converted image for malicious application flows

Fig. 2: Converted images for human, malicious application and benign application flows

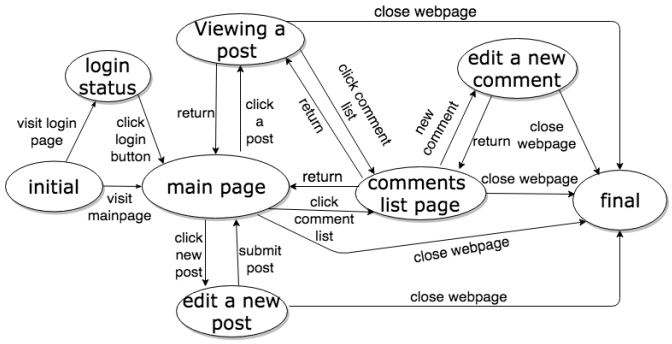


Fig. 3: How human user operations switch to each other on FSM

and collect, respectively, the flows from malicious OSN application behaviors, benign OSN application behaviors, and human user operations on OSN on the client.

B. Flow Generation

To collect flows for malicious OSN applications, benign OSN applications and human behaviors, we emulate various types of those behaviors on testbed, then collect corresponding flows.

1) Human User OSN Flows: People usually access OSNs on browsers to do various operations, such as login, post, comment, browse and so on. Their behaviors can be summarized as a series of different events. In order to simulate normal human user operations on the OSN comprehensively, we use a finite-state machine, visualized in Fig. 3 to simulate all possible human user operations on WordPress.

Human user operations on OSNs are actually driven by a series of click events on the browser. To simulate various human user operations with accurate click event intervals, the click event intervals of human operation events in the FSM follow the click pattern summarized in a real Chinese social network RenRen [24]. In this way, our collected human flows can cover all possible human user operations on this social network driven by proper human click intervals.

2) Benign OSN Application Flows: Benign API-based OSN applications use OSN released APIs to provide people useful information, such as real time weather warnings, earthquake events and news happening all around the world. In this paper, we define benign OSN application behaviors as API-based applications posting useful information on OSNs.

APIs provided by OSNs are supposed to be used for third party developers, so they can integrate OSN services to their own developed softwares, and use OSN APIs to serve people automatically when needed. Therefore, benign API-based third party applications behavior timing patterns should follow human post/comment/like patterns on OSNs. It's known that the timing pattern of when humans post/comment/like on Twitter can be modeled as different Poisson Processes [10], so benign API-based third party application behavior time points should follow Poisson Processes. Except API-based benign third party applications, there are some benign API-based bot applications on OSNs, which provide helpful weather warnings or news to people. For those benign bot applications, they will post content on a OSN whenever there is news or a warning, and those news and warnings occurrence patterns also can be modeled as Poisson Processes, so benign API-based bot applications behavior pattern can also be modeled as Poisson Processes. To sum up, all benign OSN applications behavior time points should follow Poisson Distribution.

To collect flows for all possible benign OSN application behaviors, we crawl benign posts/comments data from benign automatic accounts on Twitter, then have our benign applications posting or commenting with those benign contents, and their behavior time points are decided by various benign Poisson processes by changing different parameters.

3) Malicious OSN Application Flows: 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 only can be decided based on how people use or analyze crawled data, and this is not our focus. In this paper, we do not classify them as malicious only based on their crawling behaviors. We mainly focus on detecting malicious API-based spammer applications on OSNs. Those malicious API-based OSN applications post/comment spam contents (URLs for malwares, or spam advertisements) on OSNs.

Malicious OSN spam applications behave with malicious purposes, so their controlled spam accounts behavior patterns are probably different from benign accounts. It's been found that malicious API application controlled accounts post/retweet/like behavior time points on Twitter demonstrate five malicious patterns [10] in Fig. 4. This indicates that

corresponding malicious API applications post/retweet/like time points should also follow those five known malicious patterns. For malicious patterns in Fig. 4, x axis indicates minutes of the hour for an account post/comment/like/retweet behavior time points, while y axis indicates seconds of the minute for this account post/comment/like/retweet behavior time points.

To simulate various malicious OSN application behaviors, we implement five malicious OSN applications to post/comment spam for a single or multiple accounts. Those malicious applications behavior time points are decided by five known malicious timing patterns, and they post or comment malicious contents at each of their behavior time points. The malicious contents posted are downloaded from a Twitter dataset which provides contents posted or commented by malicious application controlled spam accounts.

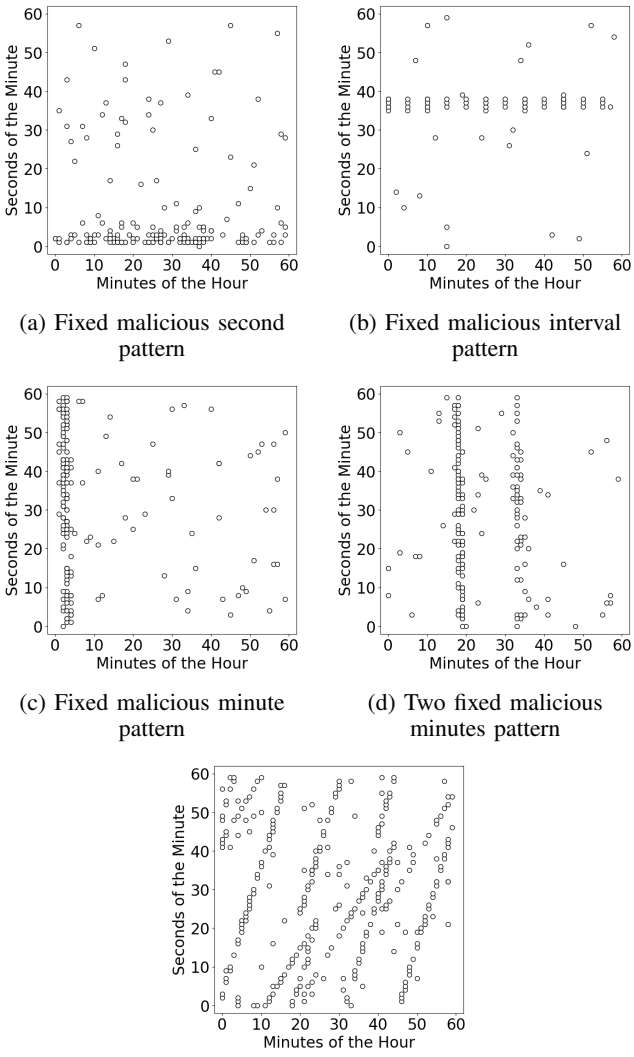
We model each malicious timing pattern as combination of different probability distributions, and simulate each malicious timing pattern by changing related parameters in corresponding probability distribution. We take the fixed malicious second pattern in Fig. 4a as an example to show how we model its probability density function and change related parameters to implement various timing instances for fixed malicious second pattern. Based on observations, the probability density function of the fixed malicious second pattern can be modeled as the combination of a uniform distribution (P_0) and a group exponential distributions ($\lambda e^{-\lambda t}$) with same λ value in formula 1. P_0 is the possibility density for uniform distribution, while $\lambda e^{-\lambda(t-dT*k-T_0)}$ indicates probability densities for multiple exponential distributions. dT is the time step length between every two adjacent exponential distributions, while T_0 is the initialized time step length for the first exponential distribution. The parameter M_0 adjusts the weight for all exponential distribution densities. Parameter λ decides the shape for all exponential distributions.

To simulate various timing pattern instances for the fixed malicious second pattern in Fig. 4a, we vary all related parameters λ, P_0, M_0, dT and T_0 in its density function. Fig. 5 shows two timing pattern instances.

In a similar way, the four probability density functions for the other four malicious API usage patterns are shown in formula 2, 3, 4 and 5 respectively, and we simulate numerous timing pattern instances for those four malicious patterns by varying corresponding related parameters with different values. In this way, our simulated timing pattern instances can cover many possible timing patterns for real world malicious applications. We can simulate all possible malicious application posting/commenting spam behaviors, and collect flows for those malicious application behaviors.

C. Optimizing Flow Generation

To speed up flow generation time for the above three categories, we optimize the flow generation strategy. We simulate how users behave both in user active time and user waiting time. During the user active time, users visit OSNs



(a) Fixed malicious second pattern

(b) Fixed malicious interval pattern

(c) Fixed malicious minute pattern

(d) Two fixed malicious minutes pattern

(e) Hybrid malicious pattern

Fig. 4: Malicious OSN applications controlled accounts behave time points show five malicious patterns

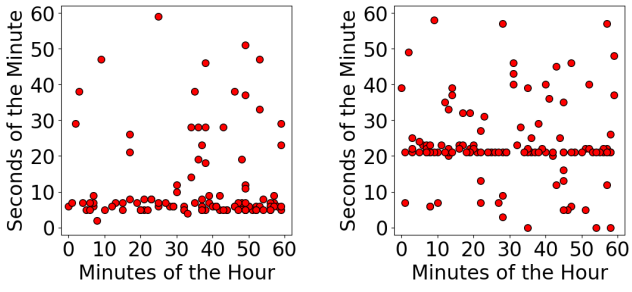


Fig. 5: Two timing instances for fixed malicious second pattern

$$f1(t,\lambda) = P_0 + M_0 \sum_{k=0}^{k=n} \lambda e^{-\lambda(t-dT*k-T_0)} \tag{1}$$

$$f2(t,\delta) = P_0 + M_0 \sum_{k=0}^{k=n} \frac{1}{\delta\sqrt{2\pi}} e^{-\frac{(t-dT*k-T_0)^2}{2\delta^2}} \tag{2}$$

$$f3(t,\lambda) = P_0 + M_0 \sum_{k=0}^{k=n} \frac{e^{-\lambda}\lambda^{(t-dT*k-T_0)}}{(t-dT*k-T_0)!} \tag{3}$$

$$f4(t,\lambda 1,\lambda 2) = P_0 + M_1 \sum_{k=0}^{k=n} \frac{e^{-\lambda_1}\lambda_1^{(t-dT_1*k-T_1)}}{(t-dT_1*k-T_1)!} + M_2 \sum_{k=0}^{k=n} \frac{e^{-\lambda_2}\lambda_2^{(t-dT_2*k-T_2)}}{(t-dT_2*k-T_2)!} \tag{4}$$

$$f5(t,\delta 1,\delta 2) = P_0 + M_1 \sum_{k=0}^{k=n} \frac{1}{\delta_1\sqrt{2\pi}} e^{-\frac{(t-dT_1*k-T_1)^2}{2\delta_1^2}} + M_2 \sum_{k=0}^{k=n} \frac{1}{\delta_2\sqrt{2\pi}} e^{-\frac{(t-dT_2*k-T_2)^2}{2\delta_2^2}} \tag{5}$$

actively, and do various operations on OSNs. During the user waiting time, users do not access OSNs and do not 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, their waiting time is much longer than their active time. Therefore, ground truth generation speed is very slow because a lot of time is wasted on simulating user waiting time.

However, during user waiting time, there are no flows generated, because users do not communicate with OSNs during waiting time. We thus simulate malicious and benign operations with short intervals, and collect flows for them, then combine flows of different OSN behaviors together, only changing each flow's start time and end time while keeping all other attributes the same. In this way, we can optimize flow generation time, while all flows still preserve their original attributes.

V. Evaluation

Based on flow data generated in section IV, we implement the solution and obtain the trained CNN model to detect flows from malicious OSN applications in section III. In this section, we first introduce the dataset size and evaluation metrics, then evaluate the performance of the trained CNN model on the test set. Last, we evaluate the model's performance by detecting flows generated by three real world malicious OSN applications and three benign OSN applications.

A. Dataset Size and Evaluation Metrics

In data generation section, we generate 40,000 aggregated flows for human user operations, benign OSN applications and malicious OSN applications in TABLE I. The dataset is split into training set and test set. Training set has 80% of all labeled aggregated data, while test set includes 20% labeled data. Training set data are used to train CNN model, while test data are to evaluate the performance of trained model. To evaluate performance of trained CNN model, we adopt four most commonly used metrics, accuracy, recall, precision and f1-measure, to evaluate the performance of our trained CNN model. Accuracy is the proportion of all predictions that are correct. Recall is a measurement of how many actual positive observations are predicted correctly. Precision measures how many positive predictions are actual positive observations. F1-Measure is the harmonic of precision and recall.

TABLE I: 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

B. Detection Performance on Test Set

To begin with, we evaluate the overall detection performance of CNN model on the whole test set. The overall performance of our model is satisfactory. It can achieve high scores for accuracy, precision, recall and f1-measure with the value of 0.976,0.963, 0.984, and 0.974 separately, and those four metrics can achieve high scores at the same time.

In this situation, we would like to check if those malicious applications post/comment with different frequencies, whether trained model can detect flows effectively. In Fig. 6, x axis indicates malicious OSN applications behavior frequencies varying from 10 times to 50 times a day, y axis shows corresponding detection results. As we can see, when malicious applications post less frequently, the detection accuracy is relatively low. The accuracy is 89.9% when malicious applications post 10 times a day. When malicious patterns post/comment more frequently, our model detection accuracy is increasing, and it reaches to 99.3% when malicious applications post 50 times.

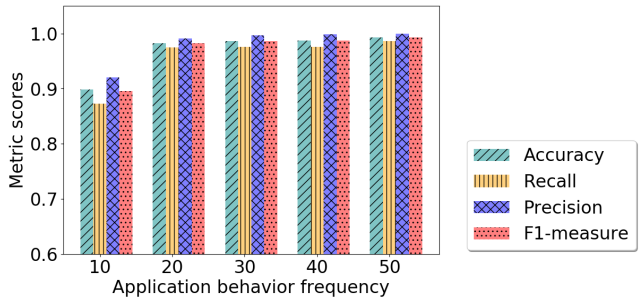
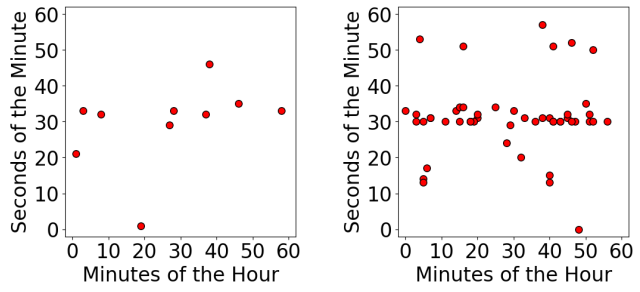


Fig. 6: Detection performance for malicious applications with different frequency

When malicious OSN applications behave more frequently, their malicious patterns are becoming obvious, then their flows are easier to be detected. Fig. 7 compares two examples for a malicious application with different behavior frequencies. In Fig. 7a, a malicious application with fixed malicious second pattern posts 10 times, and its malicious pattern is not obvious. In Fig. 7b, this malicious pattern is obvious when it posts 50 times. Therefore, when malicious applications post frequency increases, their malicious patterns are more obvious, and model can detect flows from frequent behavior applications more efficiently.



(a) Application posts 10 times (b) Application posts 50 times

Fig. 7: Comparison for fixed malicious second pattern based application posting 10 times and 50 times

Malicious applications behavior time points show five patterns in Fig. 4. To investigate whether our trained model can detect flows generated by applications with each malicious pattern effectively when their behavior frequency changes, we display a group of detection results for applications with each malicious pattern behavior frequency changing from 10 times to 50 times a day in Fig. 8. The detection results show two patterns. In Fig 8a and Fig 8c, when applications post times is less frequent, the detection accuracy is low. When applications post times is becoming more frequent, the detection accuracy is increasing until high. However, in Fig 8b, Fig 8d and Fig 8e, the detection accuracy is always high.

The reason why those results show two different patterns are related to how different malicious applications post. Let's take fixed malicious second pattern and fixed malicious interval pattern as examples in Fig. 9. For application with fixed malicious second pattern, it posts with relative long intervals. When it posts less, e.g. 10 times, its pattern is not obvious, so the detection accuracy is low. For application fixed malicious interval pattern, it posts a lot in a short time period. Even if it posts 10 times, its malicious pattern is obvious, so our detection accuracy is always high.

We also find another phenomenon: in Fig 8a and Fig 8c, when applications post less frequently, even if the detection accuracy scores are low, the precision scores are still high. Precision a measurement of how many positive predictions are actual positive observations. We represent malicious OSN application flows as positive. This high precision result indicates that our model's predicted malicious flows are likely to be actual malicious flows. When the behavior pattern of some malicious OSN applications is not obvious, our model can mistakenly predict those malicious flows as benign.

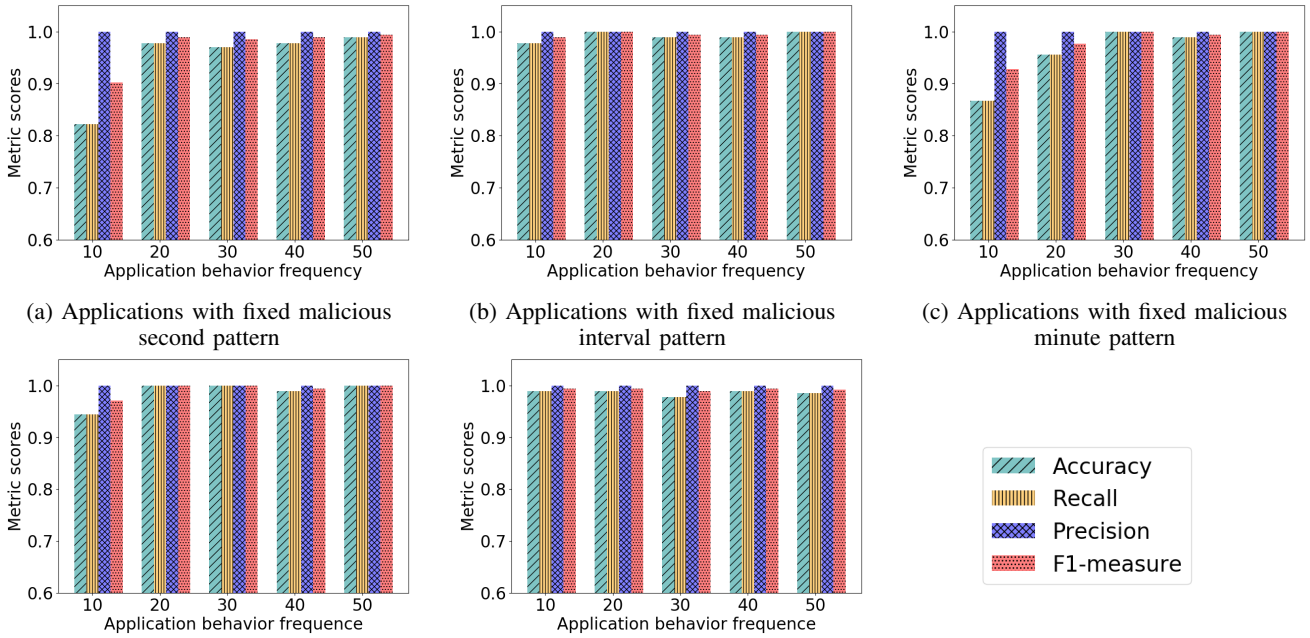
C. Detection Performance for Real World OSN Applications

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

The three real world benign OSN applications are a real time earthquakes bot, a news bot providing important news to people, and a weather warning bot respectively. The three malicious OSN applications are all spam applications. Fig. 11 shows model performance for detecting flows from three malicious OSN applications. Our detection model performs well and can detect malicious flows with accuracy as high as 99.7%, 98.8% and 99.1%. Fig. 10 shows the performance for detecting flows from three benign OSN application. Our detection model 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 can not be calculated because of no true positive and false negative, and all samples are correctly predicted as true negative or mistakenly predicted as false positive. The two figures show that our model can detect flows from real world benign and malicious OSN applications well.

VI. 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 work mainly studies how NSPs may apply a deep learning methodology 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.



(a) Applications with fixed malicious second pattern

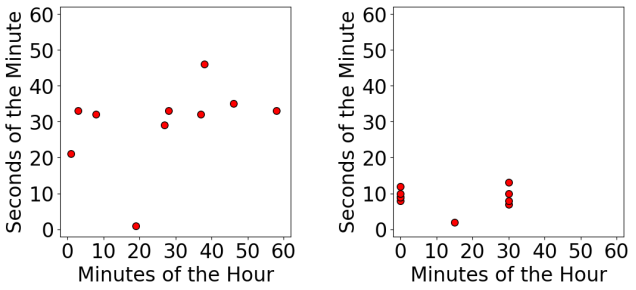
(b) Applications with fixed malicious interval pattern

(c) Applications with fixed malicious minute pattern

(d) Applications with two fixed malicious minutes pattern

(e) Applications with hybrid malicious pattern

Fig. 8: Detection results for each malicious pattern posting/commenting frequency changing from 10 times to 50 times a day



(a) Application with fixed malicious second pattern

(b) Application with fixed malicious interval pattern

Fig. 9: Comparison for fixed malicious second application and fixed malicious interval application posting 10 times

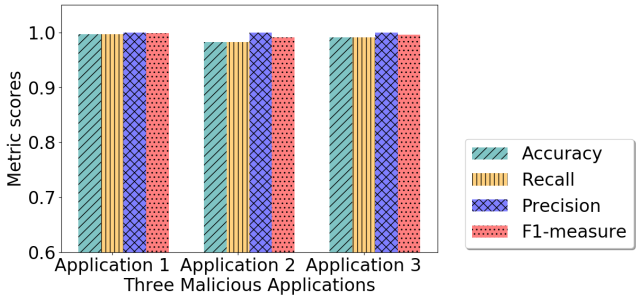


Fig. 10: Detection results for flows generated by 3 real world malicious API-based OSN applications

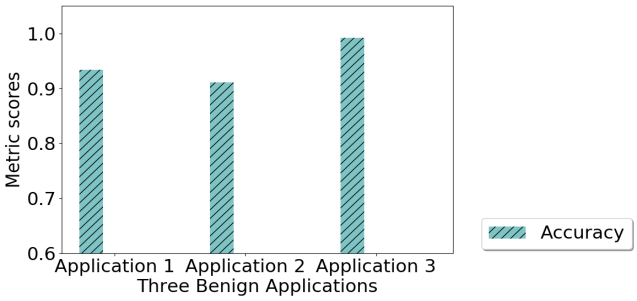


Fig. 11: Detection results for flows generated by 3 real world benign API-based OSN applications

REFERENCES

[1] N. C. Matthew Rosenberg and C. Cadwalladr, "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

[2] X. Zheng, Z. Zeng, Z. Chen, Y. Yu, and C. Rong, "Detecting spammers on social networks," *Neurocomputing*, vol. 159, pp. 27–34, 2015.

[3] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on Twitter," in *Collaboration, Electronic messaging, Anti-abuse and Spam Conference (CEAS)*, vol. 6, 2010.

[4] A. Almaatouq, E. Shmueli, M. Nouh, A. Alabdulkareem, V. K. Singh, M. Alsaleh, A. Alarifi, A. Alfaris *et al.*, "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*, vol. 15, pp. 475–491, 2016.

[5] M. Fazil and M. Abulaish, "A hybrid approach for detecting automated spammers in Twitter," *IEEE Transactions on Information Forensics and Security*, 2018.

[6] M. Mondal, B. Viswanath, A. Clement, P. Druschel, K. P. Gummadi, A. Mislove, and A. Post, "Defending against large-scale crawls in online social networks," in *ACM Proceedings of the 8th International*

Conference on Emerging Networking Experiments and Technologies, 2012, pp. 325–336.

[7] S. H. Ahmadinejad and P. W. Fong, "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*, 2013, pp. 161–166.

[8] A. H. Wang, "Don't follow me: Spam detection in Twitter," in *IEEE Proceedings of the 2010 International Conference on Security and Cryptography (SECRYPT)*, 2010.

[9] A. Saroop and A. Karnik, "Crawlers for social networks & structural analysis of Twitter," in *IEEE 5th International Conference on Internet Multimedia Systems Architecture and Application (IMSAA)*, 2011.

[10] C. M. Zhang and V. Paxson, "Detecting and analyzing automated activity on Twitter," in *International Conference on Passive and Active Network Measurement*, 2011, pp. 102–111.

[11] J. Herrera-Joancomartí and C. Pérez-Solà, "Online social honeynets: trapping web crawlers in osn," in *International Conference on Modeling Decisions for Artificial Intelligence*, 2011.

[12] R. Singh, H. Kumar, and R. Singla, "Traffic analysis of campus network for classification of broadcast data," in *Conference on Intelligent Infrastructure in 47th Annual National Convention of Computer Society of India*, 2012, pp. 163–166.

[13] J. François, S. Wang, T. Engel *et al.*, "Bottrack: tracking botnets using netflow and pagerank," in *International Conference on Research in Networking*, 2011.

[14] A. Kind, M. P. Stoecklin, and X. Dimitropoulos, "Histogram-based traffic anomaly detection," *IEEE Transactions on Network and Service Management*, vol. 6, pp. 110–121, 2009.

[15] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001, pp. 69–73.

[16] W. Zhenqi and W. Xinyu, "Netflow based intrusion detection system," in *IEEE International Conference on MultiMedia and Information Technology (MMIT)*, 2008, pp. 825–828.

[17] R. Hofstede, V. Bartos, A. Sperotto, and A. Pras, "Towards real-time intrusion detection for netflow and ipfix," in *IEEE International Conference on Network and Service Management (CNSM)*, 2013, pp. 227–234.

[18] D. van der Steeg, R. Hofstede, A. Sperotto, and A. Pras, "Real-time ddos attack detection for cisco ios using netflow," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 972–977.

[19] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing user navigation and interactions in online social networks," *Information Sciences*, vol. 195, 2012.

[20] W. Wongyai and L. Charoenwatana, "Examining the network traffic of Facebook homepage retrieval: An end user perspective," in *IEEE International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2012, pp. 77–81.

[21] Z. Móczár and S. Molnár, "Comparative traffic analysis study of popular applications," in *Meeting of the European Network of Universities and Companies in Information and Communication Engineering*, 2011, pp. 124–133.

[22] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding online social network usage from a network perspective," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, 2009, pp. 35–48.

[23] C. Orsini, A. King, D. Giordano, V. Giotsas, and A. Dainotti, "Bgpstream: a software framework for live and historical bgp data analysis," in *ACM Proceedings of the 2016 Internet Measurement Conference*, 2016, pp. 429–444.

[24] G. Wang, X. Zhang, S. Tang, C. Wilson, H. Zheng, and B. Y. Zhao, "Clickstream user behavior models," *ACM Transactions on the Web (TWEB)*, vol. 11, 2017.