MUI-defender: CNN-Driven, Network Flow-Based Information Theft Detection for Mobile Users

Zhenyu Cheng[1,2], Xunxun Chen[1,2(✉)], Yongzheng Zhang[1,2], Shuhao Li[1,2], and Jian Xu[1,2]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{chengzhenyu,zhangyongzheng,lishuhao,xujian}@iie.ac.cn, xx-chen@139.com
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. Nowadays people save a lot of privacy information in mobile devices. These information can be theft by adversaries through suspicious apps installed in smartphones, and protecting users' privacy has become a great challenge. So developing a method to identify if there are apps thieving users' personal information in smartphones is important and necessary. Through the analysis of apps' network traffic data, we observe that general apps generate regular network flows with the users' normal operations. But information theft apps' network flows have no relationship with users' operations. In this paper we propose a model MUI-defender (Mobile Users' Information defender), which is based on analyzing the relationship between users' operation patterns and network flows with CNN (Convolutional Neural Network), can efficiently detect information theft. Because of C&C (Command-and-Control) server invalidation [33] and system version incompatibility [25], etc., most of the collected information theft apps can't run properly in reality. So we extract information theft code modules from some of these apps, and then recode and compile them into the ITM-capsule (Information Theft Modules capsule) for verification. Finally, we run the ITM-capsule and several normal apps to detect the network flows, which shows our detection model can achieve an accuracy higher than 94%. Therefore, MUI-defender is suitable for detecting the network flows of information theft.

Keywords: Information theft · Network flow · Operation pattern · CNN

1 Introduction

In recent years, smartphones are used more frequently and widely in people's daily life, and people always save lots of personal information in their mobile phones, which has been accompanied with amounts of personal information theft. In the past researches, researchers have done a mint of works to detect

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2019
Published by Springer Nature Switzerland AG 2019. All Rights Reserved
H. Gao et al. (Eds.): CollaborateCom 2018, LNICST 268, pp. 329–345, 2019.
https://doi.org/10.1007/978-3-030-12981-1_23

malwares, especially on the Android system, which becomes the main stream in the market. At present, the typical detections can be classified as signature-based methods [12], content-based methods [2], behavior-based methods [13,16], etc. Although these traditional detection techniques have been widely used in real situations to solve problems, they rarely focus on detecting information theft by network traffic. It has to be noted that some of these detection methods need a priori knowledge, and some have to root the operation system or establish a sand box [29]. For these reasons, we want to develop a flow-based method to detect information theft without any priori knowledge or deeply modifying the system.

With the popularity of multiple wireless interfaces including WiFi and 3G/4G, a smartphone usually keeps persistent Internet connectivity throughout all day. As CISCO [1] reported, mobile traffic data generated by apps had grown 18-fold over the past 5 years. Consequently, collecting and analyzing network traffic became a practicable method for detecting information theft. The analysis of network traffic has been done in traditional areas, such as in personal computers [4,5,8,18,24] and workstation servers [6,11,15]. Recently, it is also applied to mobile devices [3,27,30]. But most of them focused on the questions about how to distinguish the mobile device from the others or how to classify a user's habits of using apps. It is lack of using network traffic data to detect information theft.

In this paper, we propose MUI-defender (Mobile User Information defender) to detect the app that thieves users' private information. We find out that a certain relationship exists between network flows and the user's operation patterns. When an app steals personal information, its network flows' "shape" will be different from normal apps'. So we implement a CNN-driven, network flow-based model to detect information theft, and conduct a series of experiments to validate the capability and accuracy of our work. In general, the contributions of this paper are summarized as follows:

(1) We propose a detection model combining the user's operation patterns and the app's network flows to identify the information theft. The operation patterns include three types: Tap, Swipe and Others. One unit of network flow is defined as one communication session between a client and a server, which is composed of TCP/IP packets.
(2) We design a suitable feature vector model that can be efficiently utilized by CNN (Convolutional Neural Network) to detect information theft network flows. The vector contains the value of calculated DTW (Dynamic Time Wrapping), the weight of the operation pattern and the features extracted from traffic data.
(3) We validate our model by using the real-world apps and ITM-capsule (Information Theft Modules set). Because of C&C server invalidation and system version incompatibility, we design a simulation C&C server and extract information theft code modules to recode them into ITM-capsule for verification. And we achieve an accuracy higher than 94.23%, especially in the outgoing flow type, it is up to 97.12%.

The rest of this paper is organized as follows: In Sect. 2, we discuss some related works around our research topic. The detailed content of our model is described in Sect. 3. We present the performance of the network flows detection in Sect. 4. In Sect. 5, we discuss several limitations and future work, and then we draw conclusions of this paper in Sect. 6.

2 Related Work

Recent studies mostly focused on the suspicious app's behaviors when the app was running in the mobile device. Zhou et al. [34] created a privacy mode to empower users a flexible and fine-grained manner control in apps' information access, and users could dynamically manage the access at app's runtime. Mockdroid [7] was an Android operating system modified by AR Beresford et al., and it allowed users to give an app fake permissions to access privacy resources, after that the system would return null or unavailable information. Enck et al. [13] proposed a system called TaintDroid, which could accurately track the sensitive data and then notify the user when these data left the phone.

Indeed there have been some works which utilized traffic analysis in workstations and web browsers [17,21,32], and few works have been done in mobile devices [3,14,27]. At the first glance, it seems that the traffic analysis of smartphone apps is a simple transformation from the traditional works. While there are some similarities, such as end-to-end communication using IP addresses/ports, the nuances in traffic types and the transmission mode make the traffic analysis distinct between smartphones and traditional devices.

The network level analysis of Android malware behaviors offers complementary methods of characterizing and mitigating malwares. Neasbitt et al. [23] proposed ClickMiner, a tool that reconstructed user-browser interactions from network traces. Stöber et al. [28] extracted some side-channel features from network traffic generated by the most popular apps, such as timing and data volume. They could evaluate whether these information could be used to reliably identify a smartphone. In [20], Liberatore et al. evaluated the effectiveness of naive Bayes and Jaccard, which were based upon classification algorithms to identify encrypted HTTP streams. Conti et al. [9] designed a system that could identify the specific action by using advanced machine learning techniques, when a user was performing on her mobile apps. They further compared their solution with three state-of-the-art algorithms, and the result showed that their solution outperformed others in all of the cases [10]. Shabtai et al. [26] detected deviations of a mobile application's network behaviors, and they presented a behavior-based anomaly detection system that could protect mobile device users and cellular infrastructure companies from malicious applications.

Through summarizing the existing research works above, we know that although there are some studies focusing on network behaviors about the Android malware, it is lack of using the network traffic data to detect information theft. In this paper, we proposed a detection model to determine whether information is leaking through analyzing the relationship between apps' network flows and users' operation patterns.

3 Detecting Through Network Flows with CNN

3.1 Main Idea and Architecture

We know that adversaries always use apps to steal users' personal information, so how to capture and analyse network flows generated by the target app is the first problem. Most of apps generally communicate to severs using SSL/TLS (Secure Socket Layer and Transport Layer Security) for security, which is built on the top of the TCP/IP (Transmission Control Protocol and the Internet Protocol) suite. By using SSL/TLS, most of apps' communication contents can not be acknowledged, while some other information and features can be collected and analyzed. We use Wireshark to capture the mobile internet traffic through the hot spots created by us. Although we can obtain the network flow information as much as we can, the valuable data are just a part of the gained. Hence, we restructure the traffic data and divide them into a time series of packets for different session streams. In this paper, the "network flow" is regarded as a fundamental entity, and we define it as a sequence of TCP/IP packets ordered by time during a single TCP session.

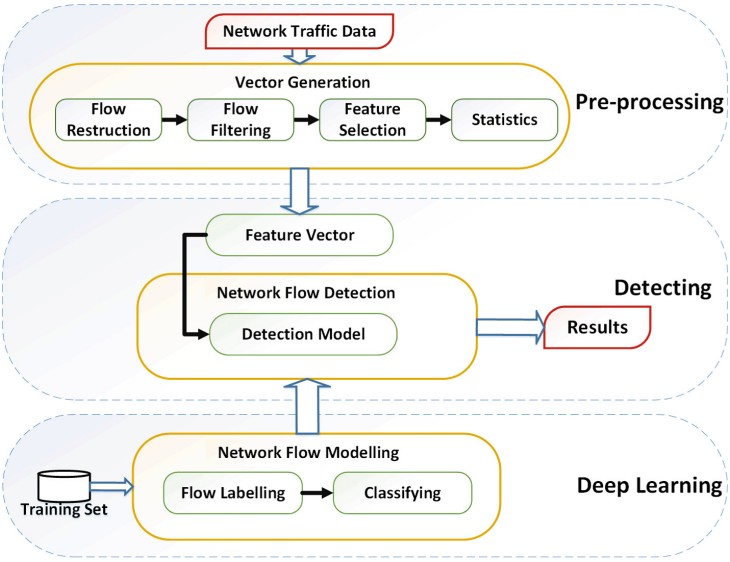


Fig. 1. MUI-defender model

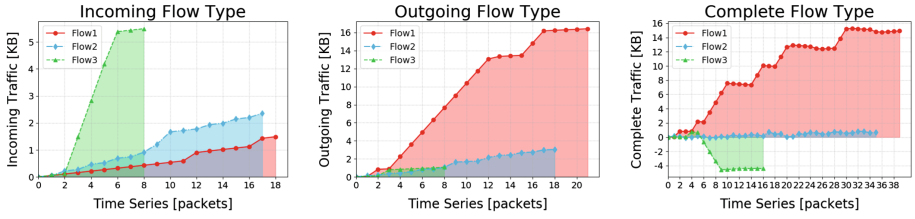
Figure 1 shows the architecture of MUI-defender, which comprises the following phases: (i) "pre-processing", which restructures and filters the captured traffic data to generate the feature vector. (ii) "deep learning", which is based on the training database to establish and train the network flow model. (iii) "detection", which has the ability to determine whether the flow is a information theft

flow or not. We detail the "pre-processing" in Sect. 3.2, the "deep learning" in Sect. 3.3 and the "detection" in Sect. 3.4.

3.2 Network Flow Pre-processing Phase

The phase "pre-processing" takes the network traffic data as inputs, and it restructures the traffic to network flows. In this phase, three types of time series are generated: (i) one is obtained by considering the bytes transported by incoming packets only; (ii) another one is obtained by considering bytes transported by outgoing packets only; (iii) a third one is obtained by combining (ordered by time) bytes transported by both incoming and outgoing packets [10]. We apply these time series as an abstract representation of communication between two endpoints.

Figure 2 depicts three examples of different apps' network flows through three cumulative graphs, which represent the three flow types. In each graph, the X-axis represents the flow time series and its unit is the number of packets. The Y-axis respectively represents the incoming traffic, outgoing traffic and complete traffic, whose unit is KB (Kilo Bytes). From these three graphs, we can recognize that the differences among the three instances' flow "shapes" are obvious. First, different flows have different lengths (packet number) in a solitary complete session. Second, the flows' content sizes have marked differences between each other. Third, the ratio of incoming traffic and outgoing traffic is distinct for the different flows. Figure 2a shows that Flow1's incoming traffic is the lowest among the three, and Flow3's is the highest one. On the contrary, Fig. 2b shows that Flow1's outgoing traffic is the highest. As Fig. 2c demonstrates that, Flow1's outgoing traffic is higher than its incoming traffic; Flow2's traffic is balanced; Flow3's incoming traffic is higher than its outgoing traffic. It should be pointed out that we consider the incoming type flow as a negative number sequence in complete series, while the outgoing type flow is considered as a sequence of positive numbers. Intuitively, we want to classify the network flows by learning their "shapes" to detect apps' information theft.



(a) Incoming (b) Outgoing (c) Complete

Fig. 2. Network flow types

After the restruction of time series, traffic data filtering is necessary to further improve the data's effectiveness and availability. Hence, a few pre-processing

steps have to be performed to generate the corresponding set of time series for each flow: (a) we use a domain filter to select the network flows that belong to the current analyzing app; (b) we continuously filter out the flows by deleting packets which could degrade the precision of our approach (i.e., we filter out ACK, SYN, FIN and retransmitted packets); (c) we limit the flow's duration or length for unified quantitative comparison. All the three pre-processing steps are essential stages before passing the network flows to the detection model. In the following, we will describe these three steps in detail.

Domain Filtering. In order to ensure that we analyze each app independently, we need to identify the traffic data for the same app. There are many different methods which can be used, and we choose the destination IP address as a parameter for the identification. However, an app may communicate with a lot of servers that have different IP addresses (e.g., the servers belong to different web services, databases and so on). To recognize all the individual IP addresses seems impossible, so we only consider the IP addresses clearly identified by their owners, and the WHOIS protocol is used in this work. We also label app's flows that we definitely know which app produces them when we catch them. In addition, we have considered network flows related to the third party services (such as Amazon, Alibaba, Unicom, etc.), which are actually used by some apps [31].

Packet Filtering. Due to network instability, network congestion, or other unpredictable situations, IP packets may be lost, delayed, duplicated, or delivered out of order. TCP discovers and solves these problems by requesting retransmission of lost or delayed packets, and reordering the packets' sequence. In this process, some TCP packets may not carry any valuable data, which can interfere with the analysis. Thus we filter out the packets which are the hinders in our research, such as duplicated packets, retransmission packets, out-of-order packets, etc. Also, the packets tagged with the ACK, SYN, FIN and RST flags are discarded, which have low value in the classification approach (i.e., we abandon the packets generated from the TCP three-way handshake in connection and four-way handshake in disconnection).

Unified Quantification. We choose two ways to quantify a network flow's size: time duration and packet number. A TCP session may contain a series of packets during the connection, so we catch the whole session's packets for further analyzing. Meanwhile, we use the constant number of packets to analyze the model of network flows. For example, in the case of a set of flow data, we select the shortest flow length as the parameter l, then we cut other flows out to pieces with the length of l. After the separation, if a flow piece's length is shorter than l, it will be discarded.

3.3 Deep Learning Phase

Labelling the network flow training data is an essential work that is used to train the models for network flows classification, and we call this stage as "Network Flow Modelling". In order to build the training dataset, we use Wireshark to

obtain traffic data through running only one app at a time. After that, we extract and label each network flow from traffic data. Two machine learning algorithms are used to build the flow models: Dynamic Time Warping (DTW) [22] and Convolutional Neural Network (CNN) [19].

Dynamic Time Warping. DTW is an algorithm used in machine learning to measure the similarity between two temporal sequences for time series analysis. DTW has been applied in many situations (e.g., video, audio, and graphic data), and actually if any data can be converted into linear sequences, DTW is able to handle them for analyzing. It is recursively defined as:

$$min\_dtw = MIN(DTW(i-1,j), DTW(i-1,j-1), DTW(i,j-1)) \quad (1)$$

$$DTW(i,j) = local\_distance(i,j) + min\_dtw \quad (2)$$

It should be noted that the implementation of the dynamic time warping is $DTW(s,t)$, while two sequences s and t are strings of discrete symbols. For two symbols i and j (i is in sequence s, j is in sequence t), the $DTW(i,j)$ is defined in terms of the shortest path up to the adjacent symbols, and the $local\_distance(i,j)$ means a distance between the two symbols (e.g., $local\_distance(i,j) = |i-j|$). The calculation of $DTW(s,t)$ is a feature included in the feature vector, which is the preparation work for the next step.

CNN. In machine learning, CNN (or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to solve some problems like image recognition. As one of the most popular deep learning algorithms, CNN overcomes the difficulties in feature extraction and is good at extracting local features. These advantages are in line with our vision for the network flow detection, so we choose CNN as our modeling basis. A CNN usually consists of an input and an output layer, as well as multiple hidden layers. The hidden layers typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers, etc. Convolutional layers apply a convolution operation to the input data, passing the result to the next layer. Pooling layers integrate the outputs of neuron clusters at one layer into a single neuron in the next layer. Fully connected layers connect every neuron in one layer to every neuron in another layer, and it is in principle the same as the traditional multi-layer perception neural networks. Figure 3 is the CNN architecture which is designed by us. We set 4 filters with the kernel size of 3, and use the "sigmod" function to squash the single-unit output layer. After training, we take the number 0 as the label of normal flows, and the number 1 as the label of information theft flows.

3.4 Detecting Phase

In the detection phase, we establish a model that uses CNN to learn and classify the network flows. It can determine whether an app is stealing users' information. To better classify network flows with users' operation patterns, we divide them into three categories: Tap, Swipe and Others. The action of Tap is the most

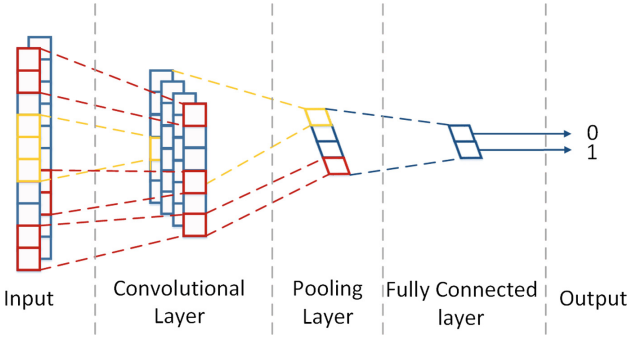


Fig. 3. The architecture of CNN

common operation that people use smartphones (e.g., a user has to tap on the app's icon to start the program). The Swipe is a type of actions by moving the finger on the screen up and down or left and right, and these actions usually happen in browsing web pages or social network information flows. The third type of actions is defined as that happens in other situations, for example, there are no actions on a smartphone (e.g., a user is sleeping without using the phone). In fact, although no actions, some network flows are still generated. Because apps can run in the system's background, and malwares prefer this way to steal information without users' permissions.

Due to the feature vectors generated by pre-processing phase and the flow models generated by deep learning phase, the detection phase could achieve an accurate classification of normal network flows and information theft ones. To better determine the user's operation pattern's influence, we set weights for flows generated by the three categories previously mentioned. Weights could be assigned to grant more importance to some types of flows against others. For example, it is comprehensible to give more weights to the flow produced by the "Others" operation type, while a user almost have no operations on the smartphone. And we give less weights to the flow produced by the Swipe, which indicates that maybe a user is surfing the Internet. At last, we use the CNN to classify network flows. We introduce the detection algorithm which outlined in Algorithm 1. The main idea is that different actions occur in different apps, which generate different classes of network flows, and the algorithm is able to correctly determine the labels of classes for unknown instances.

4 Experiment and Evaluation

In order to assess the performance of our experiment, we also consider several widespread apps in China that have different purposes: QQ, WeChat, Weibo, TouTiao, Evernote, QQMail and BaiduYun. QQ and WeChat are the most popular social network communication apps in China, which are produced by Tencent Inc., and people use them to send messages, photographs and voice messages,

Algorithm 1. Detection of network flows

Require:
Training network flows dataset: $F = \{F_0, F_1, ..., F_n\}$
Test network flows dataset: $T = \{T_0, T_1, ..., T_n\}$
Leader flows: $L = \{L_0, L_1, ..., L_7\}$
Weight set of use's operation patterns: $W = \{W_0, W_1, ..., W_n\}$
Ensure:
Classified flows: $C = \{C_{Normal}, C_{InformationTheft}\}$
for $i = 0$ to n **do**
  pre-processing F_i
  getting feature vector V_i
  put W_i into V_i {W_i is weight of behavior pattern that generates F_i}
  for $j = 0$ to 7 **do**
    $DTW(F_i, L_j)$
  end for
  put MIN(DTW) into V_i
end for
Normalization F {Make the model of CNN}
Convolution1D {which will learn 4 filters with kernel size 3}
GlobalMaxPooling1D
Add a vanilla hidden layer {Dense(),Dropout(),Activation('relu')}
Dense(1) {project onto a single unit output layer}
Activation('sigmod') {squash with a sigmoid}
Classifying F with CNN model into C
Detection T using CNN model with C
Classifying T into C
return C

etc. Weibo is a kind of Online Social Networks, in which people can post their statuses, pictures or other something interesting. TouTiao is a news aggregation app, and it supplies personalized information for users and provides new services for connecting people and information. Evernote is an app designed for note-taking and archiving. QQMail is one of the largest email services, and its Android app is at the top of ranking in Chinese market. BaiduYun is a widely used cloud storage service owned by Baidu Inc. All the selected apps use a back-end service to implement the program function, thus they must generate network traffic.

4.1 Hardware and ITM-capsule Configuration

For the assessment of our analysis, we used a Galaxy Note 4 (SM-N9100) smartphone, running the Android 6.0.1 operation system. We used a server computer (Intel Core i7-4790 3.60 GHz with 8 GB DDR3 RAM) to run Windows 7 with two network cards, which simulated a router for receiving network traffic. When communicating with smartphones, the server computer ran Wireshark to catch the traffic packets. And then we created a comma separated files (csv) from Wireshark capture files, in which each row represented a packet. The packet

reported time in seconds from Unix epoch, source and destination IP addresses, ports, size in bytes, protocol type, TCP/IP flags and some other information. Because we did not care about the payload, we discarded this part of information. These collected data were used to generate the network flows explained in Sect. 3.2.

In associate with CNCERT/CC (National Computer network Emergency Response technical Team/Coordination Center of China), many information theft app samples were provided for the analysis. In real usage scenarios, we did a lot of work to test and analyze these apps, and we found that almost all of them could not run properly. Some of these apps' C&C servers could not be connected. Some apps could not communicate with them in the original program logic. And some could not be installed on the smartphone because of operating system version incompatibility. In other words, the real information theft apps' life cycles were too short to implement the whole attacking process for analyzing. Therefore, we extracted the information theft modules from these apps and recoded them into ITM-capsule for verification, and designed a simulation C&C server to response the communication requests. ITM-capsule tried to collect the user's information such as the phone identification information, contacts list, call history, short messages, location and Internet records.

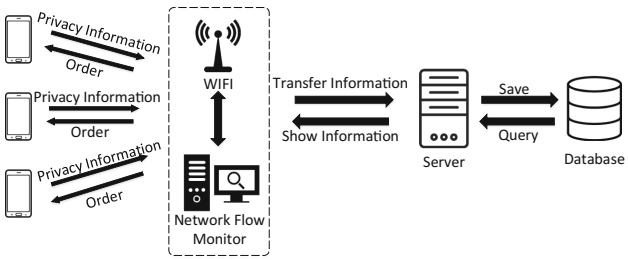


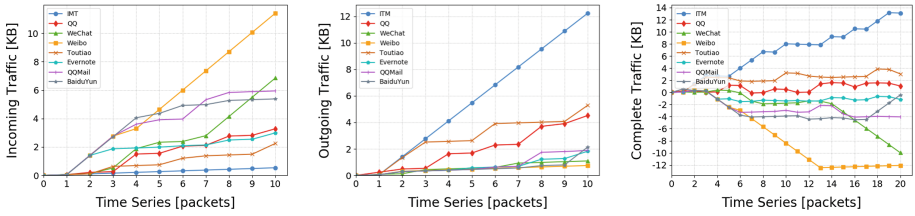
Fig. 4. The process of ITM-capsule running

Figure 4 represents the process of ITM-capsule's communication between smartphones and the server, which is supervised by our network flow monitor. ITM-capsule runs in the Android system, and it uses the HTTP (HyperText Transfer Protocol) to transfer the smartphone user's information. Firstly, it collects phone basic information, and sends the information to the server. Immediately, the server assigns an ID to the smartphone to identify the device, and sends the ID back. Secondly, ITM-capsule adds the ID to all the next transferred contents to ensure the information unity. Then it combines the phone ID with contacts, call history, SMS and so on, to deliver them separately in each new HTTP connection. Finally, when the whole transmission procedure ends, i.e., when it has not receive new data for a long time, the server sends a stop message to the smartphone to finish ITM-capsule's work.

4.2 Operation Pattern and Network Flow Analysis

To analyze the network flows, we installed seven apps from the official market: QQ v6.6.9.3060, WeChat v6.5.6, Weibo v7.3.0, TouTiao v5.3.7, Evernote v7.9.7.1079770, QQMail v5.2.7 and BaiduYun v7.17.0. For each app, we recruited two experimental volunteer and created a new account for each of them. The volunteers simulated the behaviors of users in using the apps, like sending messages, posting statuses, browsing news, etc. To deal with the social network apps, we set several friends or followers for each account. By this way, we avoided configuring the accounts with actual friends or followers, so that we could make sure the experimental environment is pure and under our control.

In order to obtain operation patterns, we developed an app called ActionCatcher that could capture and classify the actions as three categories, which is mentioned earlier in Sect. 3.4. For example, when a user wants to send messages by QQ or WeChat, he has to tap on the app's icon firstly and waits the program to connect the Internet, and then taps on a friend's avatar to open the communicating page and inputs texts or pictures. Finally, he taps on the "send" button to send messages. In the process of the above actions, ActionCatcher can match the "click" action as the Tap type. It is certain that when a user browses news or social network information flows, his finger slides on the screen and ActionCatcher can match these actions as the Swipe type. All these actions can produce network traffic data, so it is important to label the flows with the operation patterns. Since ActionCatcher records the time that actions occurred, we can match them with the network flow time. In the experiment, we limited the volunteers to operate apps in a day (24 h), and then we obtained these sets of flows, which were matched with operation patterns.



(a) Incoming Flows (b) Outgoing Flows (c) Complete Flows

Fig. 5. Apps' network flow types

As mentioned in Sect. 3.2, we could limit the length or use whole duration time of flows to achieve unified quantification. Figure 5 shows the diversity of apps' flow types which are selected with limited length (fixed number of packets). In Fig. 5a, we can find out that within 10 packets, Weibo obtains the most traffic data while ITM-capsule obtains the least. Figure 5b shows that ITM-capsule sends out the most traffic data within 10 packets, and other apps relatively are much less than ITM-capsule, especially Weibo sends the least data. Figure 5c

demonstrates the diversity of the complete flow types. From this figure we can find that ITM-capsule's outgoing flow is dominant, and Weibo's incoming flow takes over the majority of traffic data. From each subfigure, we can realize that different apps generate different flow types and their features are different. As a result, we extract the feature vector that contains: the value of DTW, operation pattern with weights, duration, and number of packets, flow length, average packet length, average packet interval for each complete flow, incoming flow and outgoing flow.

To reduce the computation burden of calculating DTW, a leader flow is elected for each app, and these leader flows are representative for their flow type. Giving a class F of flows for one app, which contains the flows $\{f_1, ..., f_n\}$, the leader is the flow f_i that is selected with the minimum overall distance from the other flows of the class, and this can be expressed as a formula:

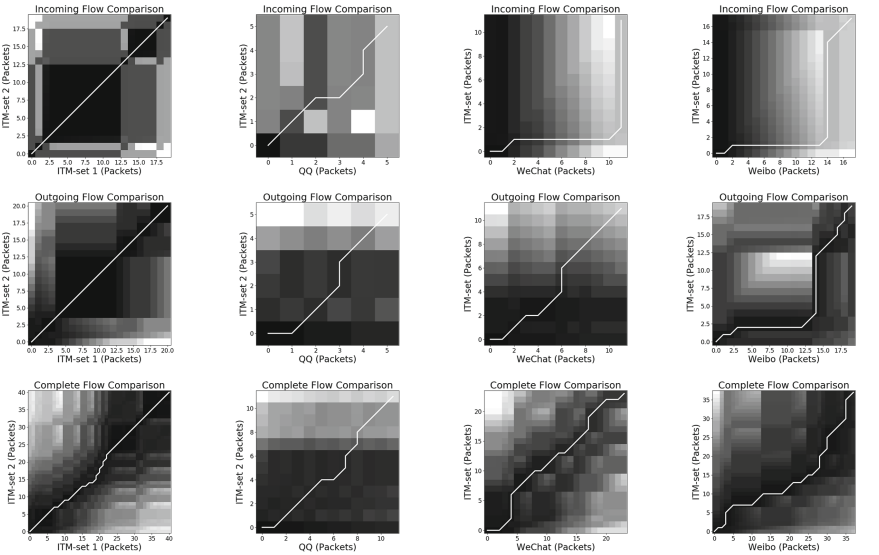
$$arg\ MIN_{f_i \in F} \left(\sum_{j=1}^{n} DTW(f_i, f_j) \right) \tag{3}$$

The $DTW()$ is declared in Sect. 3.3. Figure 6 shows the comparison between ITM-capsule's network flow and that of other seven apps using the DTW algorithm. The gray-scale image means that when the white line is straighter and smoother in the diagonal, the two flows' "shapes" are more similar and the value of the DTW are smaller. Each subfigure represents ITM-capsule's flow comparison with others', and the three graphs in the subfigure represent DTW calculation of different flow types. For example, Fig. 6b shows the comparison between ITM-capsule's flow and QQ's flow in DTW: The top graph indicates the comparison of incoming flows; The middle one is depicting difference between outgoing flows; The bottom graph shows the complete flows' discrepancy. Since the white line of the Fig. 6b is so tortuous, we can realize that ITM-capsule's flow is very different with QQ's. Indeed, we discover that the flows of ITM-capsule are distinctive with others in Fig. 6.

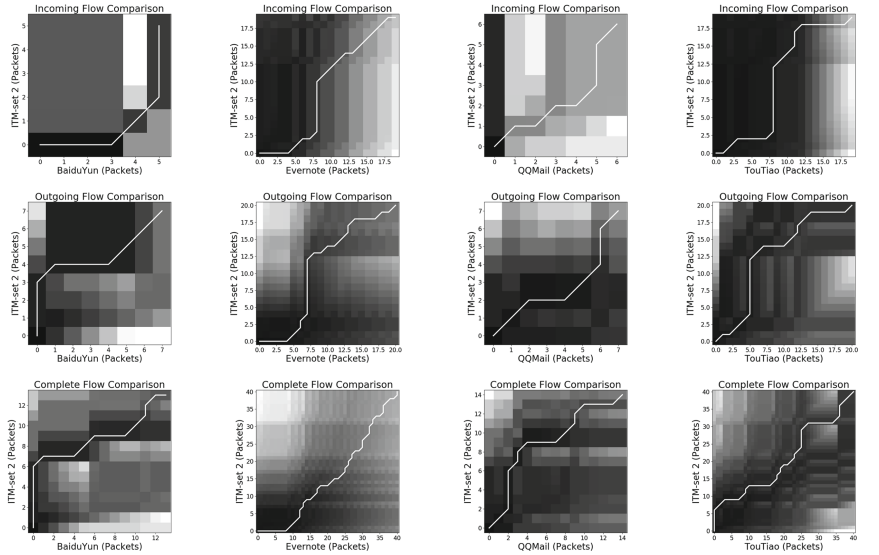
4.3 Detection Evaluation

For the network flow detection, the captured traffic data were divided into two parts: a training set and a testing set. The training set was used in machine learning phase to train the classifier, while the testing one was used to evaluate the effectiveness of our model and the accuracy of the classifier. As we created two accounts for each app to generate traffic data, we could make sure that the results of the detection do not depend on the specific accounts or users in the process of analysis.

In Table 1, we report detailed results for the accuracy, the precision, the recall, the true positive (TP) rate and the false positive (FP) rate. It demonstrates that different flow types generate different results of detection, and among these types the outgoing flow type gets the best result (97.12% accuracy, 90.45% precision and 94.34% recall). In the analysis, the FP rate of outgoing flow is a little higher



(a) ITM vs ITM (b) ITM vs QQ (c) ITM vs WeChat (d) ITM vs Weibo



(e) ITM vs BaiduYun (f) ITM vs Evernote (g) ITM vs QQMail (h) ITM vs TouTiao

Fig. 6. DTW comparison

than others, while it indicates the model is sensitive about unnormal outgoing flows. This table demonstrates our detection model is in line with the expected results, and the performance is excellent on the network flows detection with CNN.

Table 1. Detection performance

Flow type	Accuracy	Precision	Recall	TP rate	FP rate
Complete	0.9423	0.9213	0.9308	0.9321	0.0842
Incoming	0.9552	0.9249	0.9267	0.9479	0.0768
Outgoing	0.9712	0.9045	0.9434	0.9661	0.1021

5 Discussion

In this paper, we extract and recode information theft code modules into ITM-capsule, and then we use ITM-capsule and several general apps to implement the information theft detection with network flows. The result of the detection demonstrates that the distinction of outgoing flow types between general apps and information theft apps is obvious. If there are some information theft apps could run successfully in reality, we really prefer to use them to verify our detection model. As we mainly focus on the detection of users' information leakage, we do not pay attention to some other classes of malicious flows, for example, the flow data are generated by a suspicious app which is downloading malwares. And considering the convenience of management and the openness of operating system, we only verify our model in Android platform. We need to strengthen the comprehensiveness of our detection model, such as extend its detecting species and system diversity.

Because the detection is conducted after the deep learning, our analysis is inefficient for the detection in real-time, and the timeliness needs to be improved. Although we have achieved a high precision and accuracy, the detection is limited to two classes: normal flows and information theft flows. We plan to investigate more specific classes of network flows, such as flows generated by charges-consuming malwares, remote-controlling malwares, malicious-deducting malwares, etc.

We know that a network flow does not continuously carry valuable data in the packets, and the network flows generated from the same app usually have similar feature parameters. For these reasons, we select CNN for modeling, whose major advantages are local perception and parameter sharing. And it is proved that CNN is suitable for our detection. However, it is lack of comparison with other machine learning methods in this paper, we will perfect this work in the future. In our opinion, the automatic detection needs to be implemented, but CNN needs a training dataset that has to be labeled by people, and automation becomes a limitation of our approach.

6 Conclusion

In this work, we detect network flows to identify the information theft app without deeply modifying the user's mobile device. The model proposed in this paper is able to classify sets of network flows into normal or information theft flows

by using the CNN algorithm. As a popular deep learning algorithm, CNN is usually used to analyze multidimensional data, like graphics, audio, etc. But network flows are a kind of one dimension vector data, we modify the input format and the CNN's structure to fit MUI-defender. The result demonstrates that MUI-defender is effective in detecting the network flows generated by information theft apps. In addition, we evaluate the detection model with a series of traffic data, and it shows that our model achieves high accuracy (97.12%), precision (92.13%), and recall (94.34%), etc. These experimental evidences indicate that our model is both efficient and effective to apply in actual detection environments.

Acknowledgement. This work was supported by the National Key Research and Development Program of China (No. 2016YFB0801502), and the National Natural Science Foundation of China (Grant No. U1736218). We are grateful for the assistance from the volunteers. Thanks to their valuable contribution to the experiments in this paper. We also want to thank the reviewers for the thorough comments and helpful suggestions.

References

1. Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper. https://goo.gl/ylTuVx. Accessed 28 Mar 2017
2. Arzt, S., et al.: Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. ACM Sigplan Not. **49**(6), 259–269 (2014)
3. Atkins, J.B., Dobson, R.W.A.: Monitoring system for a mobile communication network for traffic analysis using a hierarchical approach. US Patent 7,830,812, 9 Nov 2010
4. Barford, P., Kline, J., Plonka, D., Ron, A.: A signal analysis of network traffic anomalies. In: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement, pp. 71–82. ACM (2002)
5. Barford, P., Plonka, D.: Characteristics of network traffic flow anomalies. In: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, pp. 69–73. ACM (2001)
6. Benson, T., Akella, A., Maltz, D.A.: Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, pp. 267–280. ACM (2010)
7. Beresford, A.R., Rice, A., Skehin, N., Sohan, R.: Mockdroid: trading privacy for application functionality on smartphones. In: Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, pp. 49–54. ACM (2011)
8. Chandra, R.: Network traffic monitoring for search popularity analysis. US Patent 7,594,011, 22 Sept 2009
9. Conti, M., Mancini, L.V., Spolaor, R., Verde, N.V.: Can't you hear me knocking: identification of user actions on Android apps via traffic analysis. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, pp. 297–304. ACM (2015)
10. Conti, M., Mancini, L.V., Spolaor, R., Verde, N.V.: Analyzing Android encrypted network traffic to identify user actions. IEEE Trans. Inf. Forensics Secur. **11**(1), 114–125 (2016)

11. Deng, J., Han, R., Mishra, S.: Decorrelating wireless sensor network traffic to inhibit traffic analysis attacks. Perv. Mob. Comput. **2**(2), 159–186 (2006)
12. Desnos, A., et al.: Androguard: Reverse engineering, malware and goodware analysis of android applications (2013). google.com/p/androguard
13. Enck, W., et al.: Taintdroid: an information flow tracking system for real-time privacy monitoring on smartphones. Commun. ACM **57**(3), 99–106 (2014)
14. Falaki, H., Lymberopoulos, D., Mahajan, R., Kandula, S., Estrin, D.: A first look at traffic on smartphones. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, pp. 281–287. ACM (2010)
15. Fusco, F., Deri, L.: High speed network traffic analysis with commodity multi-core systems. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, pp. 218–224. ACM (2010)
16. Grace, M., Zhou, Y., Zhang, Q., Zou, S., Jiang, X.: RiskRanker: scalable and accurate zero-day android malware detection. In: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, pp. 281–294. ACM (2012)
17. Hintz, A.: Fingerprinting websites using traffic analysis. In: Dingledine, R., Syverson, P. (eds.) PET 2002. LNCS, vol. 2482, pp. 171–178. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36467-6_13
18. Lakhina, A., Papagiannaki, K., Crovella, M., Diot, C., Kolaczyk, E.D., Taft, N.: Structural analysis of network traffic flows. In: ACM SIGMETRICS Performance Evaluation Review, vol. 32, pp. 61–72. ACM (2004)
19. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
20. Liberatore, M., Levine, B.N.: Inferring the source of encrypted HTTP connections. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 255–263. ACM (2006)
21. Mah, B.A.: An empirical model of HTTP network traffic. In: Proceedings of Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution, INFOCOM 1997, vol. 2, pp. 592–600. IEEE (1997)
22. Müller, M.: Information Retrieval for Music and Motion, vol. 2. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74048-3
23. Neasbitt, C., Perdisci, R., Li, K., Nelms, T.: ClickMiner: towards forensic reconstruction of user-browser interactions from network traces. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1244–1255. ACM (2014)
24. Qadeer, M.A., Iqbal, A., Zahid, M., Siddiqui, M.R.: Network traffic analysis and intrusion detection using packet sniffer. In: Second International Conference on Communication Software and Networks, ICCSN 2010, pp. 313–317. IEEE (2010)
25. Schmidt, A.D., et al.: Enhancing security of Linux-based Android devices. In: Proceedings of 15th International Linux Kongress. Lehmann (2008)
26. Shabtai, A., Tenenboim-Chekina, L., Mimran, D., Rokach, L., Shapira, B., Elovici, Y.: Mobile malware detection through analysis of deviations in application network behavior. Comput. Secur. **43**, 1–18 (2014)
27. Sommer, C., German, R., Dressler, F.: Bidirectionally coupled network and road traffic simulation for improved IVC analysis. IEEE Trans. Mob. Comput. **10**(1), 3–15 (2011)

28. Stöber, T., Frank, M., Schmitt, J., Martinovic, I.: Who do you sync you are?: smartphone fingerprinting via application behaviour. In: Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 7–12. ACM (2013)
29. Tan, D.J., Chua, T.W., Thing, V.L., et al.: Securing Android: a survey, taxonomy, and challenges. ACM Comput. Surv. (CSUR) **47**(4), 58 (2015)
30. Tang, D., Baker, M.: Analysis of a local-area wireless network. In: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, pp. 1–10. ACM (2000)
31. Wei, X., Gomez, L., Neamtiu, I., Faloutsos, M.: Profiledroid: multi-layer profiling of android applications. In: Proceedings of the 18th Annual International Conference on Mobile Computing and Networking, pp. 137–148. ACM (2012)
32. Yu, S., Zhao, G., Dou, W., James, S.: Predicted packet padding for anonymous web browsing against traffic analysis attacks. IEEE Trans. Inf. Forensics Secur. **7**(4), 1381–1393 (2012)
33. Zhou, Y., Jiang, X.: Dissecting android malware: characterization and evolution. In: 2012 IEEE Symposium on Security and Privacy (SP), pp. 95–109. IEEE (2012)
34. Zhou, Y., Zhang, X., Jiang, X., Freeh, V.W.: Taming information-stealing smartphone applications (on Android). In: McCune, J.M., Balacheff, B., Perrig, A., Sadeghi, A.-R., Sasse, A., Beres, Y. (eds.) Trust 2011. LNCS, vol. 6740, pp. 93–107. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21599-5_7