I Know Your Social Network Accounts: A Novel Attack Architecture for Device-Identity Association

Yinhao Xiao, Yizhen Jia, Xiuzhen Cheng, Shengling Wang, Jian Mao, and Zhenkai Liang

Abstract—Online social networks revolutionize the way people interact with each other. When various social network information aggregates over time, a rich online profile of the user is formed. Owing to the various features provided by mobile devices, a user's online social activities are tightly tied to his phone, and are conveniently, sometimes unnecessarily, available to social networks. In this article, we propose a novel attack architecture to show that attackers can infer a user's social network identities behind a mobile device through new dimensions. Specifically, we first developed a correlation between a user's device system states and the social network events, which leverage multiple mechanisms such as the learning-based memory regression model, to infer the possible accounts of the user in the social network app. Then we exploited the social network to social network correlation, via which we correlated information across different social networks, to identify the accounts of the target user. We implemented and evaluated these attacks on three popular social networks, and the results corroborate the effectiveness of our design.

1 Introduction

Online social networks such as Twitter and Flickr revolutionize the ways people interact with each other. Users write posts to the public and share information with family and friends about many aspects of their lives. When the social network information aggregates over time, it forms a rich online profile of the users, containing information about their location, activity, thoughts, and other aspects of daily lives. Though the sensitive information of the profile is protected by privacy settings of the corresponding social networks and is not publicly available, the aggregation of non-sensitive information may lead to a significant revelation of user privacy.

- *Yinhao Xiao is with the School of Information Science, Guangdong University of Finance and Economics, Guangzhou, Guangdong 510320, China. E-mail: 20191081@gdufe.edu.cn.*
- *Yizhen Jia is with the Department of Computer Science, The George Washington University, Washington, DC 20052 USA. E-mail: chen2015@gwu.edu.*
- *Xiuzhen Cheng is with the School of Computer Science, Shandong University, Qingdao Shandong266237, China. E-mail: xzcheng@sdu.edu.cn.*
- *Shengling Wang is with the College of Information Science and Technology, Beijing Normal University, Beijing 100875, China. E-mail: wangshengling@bnu.edu.cn.*
- *Jian Mao is with the School of Electronic and Information Engineering, Beihang University, Beijing 100190, China. E-mail: maojian@buaa.edu.cn.*
- *Zhenkai Liang is with the School of Computing, National University of Singapore, Singapore 119077. E-mail: liangzk@comp.nus.edu.sg.*

Manuscript received 7 Oct. 2018; revised 11 Sept. 2020; accepted 25 Jan. 2022. Date of publication 1 Feb. 2022; date of current version 14 Mar. 2023.
This work was supported in part by the National Science Foundation of the U.S. under Grant CNS-1704397, in part by the National Natural Science Foundation of China under Grants 62002067, 61772080, 62072044, and 72101059, and in part by the Natural Science Foundation of Guangdong Province under Grant 2020A1515010924.
(Corresponding author: Xiuzhen Cheng.)
Recommended for acceptance by A. Squicciarini.
Digital Object Identifier no. 10.1109/TDSC.2022.3147785

Researchers have demonstrated that the structural similarity of social network nodes, which is derived from users' friending information, can be used to associate accounts of the same user across different social networks, and thus de-anonymize user identity [1], [2], [3], [4]. These attacks, however, can only de-anonymize a portion of the users in a large social network structure and may not be able to de-anonymize a particular individual. On the other hand, more and more users begin to access social networks through mobile devices, making it completely possible to de-anonymize a target individual based on the side-channel information from his mobile device due to the rich features generated by the user's device while operating on social network apps. We call this kind of attack *device-identity association*. Such an attack can be highly pernicious and possibly lead to devastating economic damage: it was reported that approximately the real identities of 1.2 billion social network accounts had been leaked by the end of 2019 [5], with an average financial loss of $90 to $305 per identity [6].

Device-identity association requires collecting side-channel information from victim devices. Though it is relatively difficult for an attacker to trick users into installing a malware with strong permissions, recent research shows that public information obtained from the Android system without permission can be used to link to a user's identity [7]. Nevertheless, the attack method proposed in [7] by observing the fixed TCP payload sequence pattern is no longer effective due to the recent updates of the social network apps. We manually monitored the Twitter app's TCP sequence while tweeting and found that the sequence was irregular and noisy. Then we used Frida, an injection framework [8], to trace the function calls of the Twitter app, and found that the root cause lies in that the Twitter app no longer handles each tweeting event in a separate TLS session (separate calls to the `url.openConnection()` function), but rather combining them into one

TLS session (only one call to the `url.openConnection()` function). Therefore, more recent research investigated new techniques on achieving device-identity association [9][10].

We also observed that attacks based solely on mobile device side-channels face challenges due to limited information from one social network app. Nevertheless, in practice, the majority of social network users access their accounts very infrequently, according to our study on 500,008 Twitter accounts (see Section 4); but a user typically has accounts in different social networks such as Twitter, Flickr, and Instagram and accesses these accounts with the same device; furthermore, the social network accounts of the same user are often highly similar/correlated in their user profiles, e.g., user name, picture, and locations. Such facts reveal a high potential of device-identity association based on account correlations, which will be explored in our study.

The objective of this paper is to investigate efficient and effective techniques to accurately identify a target user (more precisely, the social network accounts of the user) in social networks even though the information obtained by an attacker is limited. In order to accomplish this goal, we proposed a novel attack architecture with two attack vectors, correlation from device system states to a social network (DS-SN) and correlation for cross social networks (SN-SN). For the DS-SN attack, we studied the association between a user's identity in a social network and the user's device system states, e.g., memory and network data. In our threat model, an attacker can get information from the system level of a user's smartphone through installed apps without any permission or the user's consent. Leveraging these states, the attacker can infer the system events, e.g., activity transitions and keyboard status, which can be used to further infer the user's social network events, e.g., sending tweets and posting Instagram photos at certain timestamps. The attacker then collects and aggregates these social network events to identify the target user's identity in the social network. However, DS-SN is sometimes not enough to identify a user's identity due to limited social network events the user leaves in a single social network; therefore, we proposed the idea of SN-SN attack, which exploits the cross social network similarity to allow the attacker to efficiently and accurately figure out the identities of a user through the system and network state left by the user's activities and account profiles on the social networks. The main technical challenge of employing SN-SN correlation and combining it with DS-SN is the high computational overhead involved in figuring out the best match among the large number of possibilities.

Contributions and Novelties.: The contributions and novelties of this paper are summarized as follows:

- We proposed a novel attack architecture which is composed of two attack vectors to identify a user's identity in social networks through DS-SN correlation and SN-SN correlation. Compared to the existing work, which is mainly based on structural similarity and/or attribute similarity of the social network accounts to de-anonymize a portion of the social network users, our approach incorporates the publicly available Android system data to preciously associate a victim's device with his social network accounts, i.e., precisely de-anonymize a target user.
- We developed techniques to accurately infer a user's social network activities from various device side-channel information including memory usage, CPU usage, and network data. We proposed a learning-based mechanism to tolerate the influence caused by the unrelated events, such as viewing posts and commenting, and the accuracy of event time inference can be up to 96.81%. In addition, based on our proposed mechanism, we were able to greatly narrow down the number of candidate accounts for 5 celebrities and uniquely identify the account for each of the 5 volunteers from more than 50,000 social accounts based on only three social network events, which significantly improve the efficiency and reduce the bar of device-identity correlation.
- We developed a novel profile similarity metric incorporating a new learning model to match profiles from different social networks for the SN-SN attack. Our method can easily eliminate more than 90% of incorrect matches, which can significantly enhance the correlation attack accuracy when it is hard to collect sufficient event data from a single social network.

This paper is organized as follows. Section 2 outlines the most related work. Section 3 presents our threat model, introduces the publicly available Android system states exploited by the attacker, and provides an overview of our attack architecture. Section 4 details the design of our attack architecture. Section 5 demonstrates the performance of our attacks based on real-world social network events and Section 6 concludes the paper.

2 Related Works

In this section, we summarize the most related work from two aspects: side-channel attacks in mobile systems and privacy attacks in social networks.

Side-Channel Attacks in Mobile Systems.

Based on the study by Xiao *et al.* [11], side-channel attacks in mobile systems have been constantly causing more and more concerns in recent years.

Chen *et al.* [12] managed to infer the UI state and deploy multiple attacks based on the side-channel information such as memory, CPU, and network statistical data stealthily obtained by a zero-permission malicious app residing in the victim's smartphone. Li *et al.* [13] proposed a keystroke inference attack targeting mobile devices by performing the principal component analysis (PCA) on the channel state information that could be affected by the finger motions through a public WiFi hotspot. Zhou *et al.* [7] exploited the side-channel information of Android devices to infer a user's private information, e.g., rough location, health condition, investment, and driving route. Yang *et al.* [14] presented an approach to discover which website a user is browsing by analyzing the USB power while the smartphone is charging. Song *et al.* [15] managed to crack a 3D printer by reconstructing the physical prints and their corresponding G-code through scrutinizing acoustic and magnetic information obtained from Android built-in sensors. Gruss *et al.* [16] exploited the prefetch instructions to defeat address space layout randomization (ASLR), which is a technique to make the memory address unpredictable for an attacker to launch code-reuse attacks such as return-oriented programming (ROP). Van *et al.* [17] took a step further to attack hardware by launching a row hammer attack and using timing inference to make the row hammer attack deterministic compared to prior attacks which can only succeed on a probabilistic sense. Li *et al.* [18] proposed a side-channel attack to infer the basic living activities by analyzing the changes of the traffic sizes of encrypted video streams in smart home surveillance. Yang *et al.* [19] identified and systematically analyzed a new security issue in HTML5-based hybrid mobile applications, which is termed the Origin Stripping Vulnerability (OSV), and proposed an OSV detection mechanism, namely OSV-Hunter, which leverages the `postMessage` API to defend against OSV from the root. Zhang *et al.* [20] conducted an empirical study on the problem of cross-principal manipulation (XPM) of web resources, and designed a toll named XPMChecker to automatically detect XPM. An analysis generated by XPMChecker reveals that nearly 49.2% apps from Google Play were affected by the XPM issue [20].

Zhang *et al.* [21] developed a novel smartphone-fingerprinting attack by exploiting the per-device factory calibration data of the embedded sensors. Yu *et al.* [10] stealthily extracted the network traffic information and leveraged the passively received broadcast and multicast (BC/MC) packets, combined with a new multi-view wide and deep learning (MvWDL) framework, to identify a victim's mobile device. Brennan *et al.* [22] discovered that Java virtual machine (JVM) introduces a crucial timing side channel when handling just-in-time (JIT) compilation, allowing an attacker to predict the input of a program, and causing severe sensitive information leakage in Java-based mobile systems such as Android.

Side-channel information is now an essential ingredient of the mobile system security.

Social Network Privacy. Studies showed that privacy of social networks can be breached in multiple ways. Backes *et al.* [23] proposed a novel link prediction inference attack between any pair of individuals in social networks based on their mobility profiles. Wondracek *et al.* [24] demonstrated that it is possible to de-anonymize a user on a social network through his membership in specific social network groups by exploiting the browser cache in order to detect whether a user has visited certain URLs of a group. Nilizadeh *et al.* [3] proposed a community-based large scale de-anonymization attack using structural similarity. Ji *et al.* [25] presented the seed quantification requirements for perfect de-anonymizability and partial de-anonymizability of the real-world social networks, which consolidate the de-anonymization at theory level. Srivatsa *et al.* [2], on the other hand, leveraged mobility traces with social networks as side-channel information to uncover users' real identities. Lai *et al.* [26] exploited a user's interest group information to de-anonymize the users on social networks. Existing studies also tended to reveal hidden attributes for social network users. Chaabane *et al.* [27] exploited a user's interest to explore hidden attributes of a user, e.g., age, gender, and so on. Mei *et al.* [28] reported an inference attack framework that integrates and modifies the existing state-of-the-art convolutional neural network models to infer a user's age. Gong *et al.* [29] proposed a new type of inference against user's hidden attributes such as location, occupation, and interest by analyzing both his social friending and behavioral records. Hassan *et al.* [30] analyzed the privacy threats in fitness-tracking social networks and developed an attack against Endpoint Privacy Zones (EPZs) to extract a user's sensitive locations. They also presented an EPZ fuzzing technique based on geo-indistinguishability to mitigate a user's privacy leak through fitness-tracking social networks.

Mondal *et al.* [31] designed a classifier which can identify posts with incorrect sharing settings at high accuracy, implicating that social network posts can be effectively used to de-anonymize a user. Zhang *et al.* [32][33] demonstrated that by leveraging user attributes one can significantly improve the social network de-anonymization accuracy.

Many of these attacks mainly de-anonymize users based on their social relationships in a large dataset, while our approach can precisely identify the social network accounts of a target user behind a device based on the device's side-channel information, the public social network events, and the profile similarity of the user's accounts in different social networks.

3 Overview and Background

In this section, we present our threat model, introduce the publicly available Android system states, and provide an overview on our attack architecture.

3.1 Threat Model

We consider an attacker who can accomplish inference attacks through a malicious app installed on a victim's device with no special permission – this malicious app only requests the `INTERNET` permission from Android to send the collected data to the attacker for analysis, which by default is automatically granted by the Android system without notifying the user. How to install a malicious app in a victim's device is out of the scope of this paper. Here are a few viable approaches: an attacker can trick a user to install a malicious app by simply disguising the app as a benign one and putting it on the Google Play Store since current automatic malware detection for Android is still not sufficiently reliable [34]; alternatively, an attacker can secretly install a malicious app on a victim's device by exploiting a remote code execution (RCE) vulnerability such as CVE-2017-0561, which allows RCE on Wi-Fi SoC [35]. In this paper, we assume that a malicious app has already been installed on the victim's device. This malicious app can silently collect and send to the attacker certain system states that are publicly available.

3.2 Publicly Available System States in Android

In this subsection, we present the publicly available state information in Android that are exploited by our proposed inference attacks. We focus on three common channels of public information, namely memory, CPU, and network.

3.2.1 *Memory*

The Android system leverages Android Runtime (ART) or its predecessor, Dalvik Virtual Machine (DVM), as the runtime environment to execute binaries in the Dalvik Executable (DEX) format. Both ART and DVM use the paging and memory-mapping mechanisms [36] to manage memory allocation of apps. Particularly, Android maintains four types of memory data for each process running on the

TABLE 1
Memory Size Information

Type of Memory Size	Description
Virtual Set Size (VSS)	Total virtual memory of a process
Resident Set Size (RSS)	Total physical memory of a process
Proportional Set Size (PSS)	Memory shared between a process and other processes
Unique Set Size (USS)	The set of pages unique to a process

system: `Virtual Set Size` (*VSS*), `Resident Set Size` (*RSS*), `Proportional Set Size` (*PSS*), and `Unique Set Size` (*USS*), which are listed in Table 1.

Android does not treat memory information as sensitive system data. Consequently, an app can obtain other apps' memory information without requesting any permission. There are four methods to retrieve the memory data of an app, which are listed below:

- The Android API has a standard class `Debug.MemoryInfo`, which provides complete interfaces to query the information for a process [37], including private dirty pages, shared dirty pages, and the PSS for Dalvik.
- The `/proc` file system has `/proc/pid/statm`(`pid` is the id of the process) that lists all four types of memory information: *VSS*, *USS*, *RSS*, and *PSS*.
- The commands "`top`" and "`ps`" in Android Toolbox [38] yield *VSS* and *RSS* for any given process. These commands also use information from the `/proc`file system.
- In some devices, `/system/xbin/procrank` is provided. The command `procrank` yields all four types of memory information for all processes in real time.

3.2.2 CPU Usage

Android maintains three types of CPU usage data for a process, which are publicly available:

- CPU usage rate: the total percentage of time a CPU operates on a running process, with 100% indicating that for a given period, the CPU spends all its available cycles running the specific process.
- User time (`utime`): the CPU time spent in the user code of a process, measured in clock ticks.
- System time (`stime`): the CPU time spent in the system code (which is the kernel) of a process, measured in clock ticks.

A zero-permission app can retrieve the CPU usage data of another app by either (1) accessing `/proc/pid/stat` which lists *user time*, *system time*, and other time-related information; or (2) using commands "`top`" and "`ps`" with Android Toolbox to return the user time, system time, and CPU usage percentage of any given process.

3.2.3 Network

Android does not store the contents of TCP packets of any process; but it maintains a record of the number of bytes sent and received through TCP connections by a process. This information is available to any zero-permission app, and can be obtained by the following ways:

- The `TrafficStats` from Android API has the method `getUidRxBytes(int uid)`, which returns the received bytes for a given user ID `uid`.
- The `/proc` system files `/proc/uid_stat/pid/tcp_snd` and `/proc/uid_stat/pid/tcp_rcv` respectively maintain the bytes sent and received for an app.

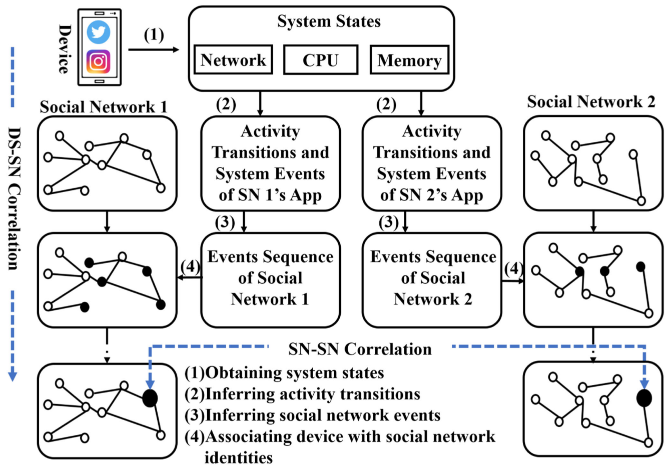
Device
(1)
System States
Network
CPU
Memory
Social Network 1
(2)
Activity Transitions and System Events of SN 1's App
(2)
Activity Transitions and System Events of SN 2's App
Social Network 2
DS-SN Correlation
(3)
(3)
(4)
Events Sequence of Social Network 1
Events Sequence of Social Network 2
(4)
SN-SN Correlation
(1)Obtaining system states
(2)Inferring activity transitions
(3)Inferring social network events
(4)Associating device with social network identities

Fig. 1. Attack architecture.

3.3 Approach Overview

The objective of our design is to associate an Android device (running social network apps to perform social activities such as posting a photo or a message) with the social network accounts (the accounts accessed by the device) based on the following publicly available information: (i) the system state information (see Section 3.2) collected by a malicious app installed in the victim Android device; and (ii) the social network events crawled from the social network databases. We consider three popular social networks: Twitter, Flickr, and Instagram, and exploit the tweeting events in Twitter and photo-posting events in Flickr and Instagram for our studies on device-identity association.

Our attack architecture is shown in Fig. 1. It is composed of two attack vectors: *device-social network correlation (DS-SN)* attack and *social network-social network correlation (SN-SN)* attack. The objective of the DS-SN attack is to identify a list of candidate accounts that might have been accessed via the device for a target social network. We developed an approach to infer a user's social network events from his device system states. More specifically, we leveraged the system states collected by a zero-permission malicious app installed in the user's device to identify the events triggered by the social network app, e.g., the time to tweet and the size of the tweet in Twitter, or the time of posting a photo in Flickr/Instagram. By correlating such information with the public events in the target social network, one can identify a list of candidate accounts in the social network for the user who may have used the device to access his account. To uniquely identify the social network account accessed via the device, one can observe multiple social network events triggered by the device to shorten

TABLE 2
Summary of the DS-SN Attack

	Inferred SN Events	Corresponding Activity Transition	Corresponding System States
Twitter	Tweeting Timestamp	`MainActivity` ⇓ `ComposerActivity`	*VSS* of Twitter `tcp_snd` of Twitter `tcp_rcv` of Twitter
	Number of Characters of a Tweet	N/A	`utime` of Keyboard `stime` of Keyboard
Instagram	Photo-posting Timestamp	`MediaCaptureActivity I` ⇓ `MediaCaptureActivity II`	*VSS* of Instagram `tcp_snd` of Instagram `tcp_rcv` of Instagram
Flickr	Photo-posting Timestamp	`MainActivity` ⇓ `FilterUploadActivity`	*VSS* of Flickr *RSS* of Media Process `tcp_snd` of Flickr `tcp_rcv` of Flickr

the candidate account list, which may take a long time if the user does not frequently access his account via the device. If the device is used to access multiple accounts of the same user belonging to different social networks, the attacker can employ the SN-SN attack that examines the profile similarity between two social network accounts from two different social networks to identify the most possible account for the user in each social network. More specifically, the attacker first obtains the candidate account lists for two (or more) social networks via the DS-SN attack, then calculate the profile similarity of two accounts, with one from each account list, and identify the pair of accounts with the highest similarity. Such a SN-SN attack can not only speedup the process of device-identity association attack, but also help identify two or more social network accounts accessed via the same device for the victim.

4 Design and Implementation of Our Attacks

In this section, we detail our design of the two inference attacks: DS-SN attack and SN-SN attack.

4.1 DS-SN Correlation Attack

Our approach to attacking the DS-SN correlation is composed of 4 steps: 1) obtaining system states from a victim's device via a zero-permission malicious app, 2) inferring activity transitions in the device, 3) inferring the corresponding social network events triggered by the device, and 4) associating the victim's device with his social network account based on the inferred events triggered by the device and the publicly available events crawled from the target social network. We considered three popular social networks: Twitter, Flickr, and Instagram. Our purpose is to infer the tweeting time of and the number of characters in a tweet for Twitter, the posting time of a photo in Instagram, and the posting time of a photo in Flickr. This inference is based solely on the publicly available information: the Android system states collected by a zero-permission app and the tweeting/photo-posting events crawled from the social network databases. The device state information needed by the DS-SN attack as well as the inferred device activity transitions and social network events for the three social networks are presented in Table 2.

4.1.1 Obtaining System States

As shown in Table 2, we exploited *VSS*, *RSS*, `utime`, `stime`, `tcp_snd`, and `tcp_rcv` for our DS-SN attack. There are multiple ways to obtain these system data in Android, as elaborated in Section 3.2. The most intuitive way is to make calls to the Android APIs. However, one has to retrieve the three types of data using different APIs, making this method less efficient. To overcome this problem, we first leveraged the "`ps`" command with Android Toolbox [38], which can return the memory and CPU data in one call; then we directly read the system files `/proc/uid_stat/pid/tcp_snd` and `/proc/uid_stat/pid/tcp_rcv` to retrieve the network data.

4.1.2 Inference of Activity Transitions

Activity transition is one of the most critical states to infer private information from an app. Previous research done by Chen *et al.* [12] described an approach that can infer an app's activity transition using a Hidden Markov model (HMM) over the memory data and then further infer the current foreground UI/activity the user is landing on. We tested this approach in our Android devices but failed to make it work. A careful study indicates that this approach is quite sensitive to the quality of the memory data but social network apps when running constantly incur significant noises compared with the clean activity transitions considered in [12]. Fig. 2 illustrates the variations of the *VSS* of the Twitter app in four Android devices with different OSes when we performed swiping and tweeting for approximately two minutes. Note that people usually swipe when tweet; thus these two activities are commonly performed together. From this study one can conclude that social network apps incur noisy memory data in most Android OSes, and the HMM inference method would face challenges when distinguishing tweeting from swiping. Having noticed this fact, we propose our own approach to precisely infer the activity transitions with the noisy memory data.

To proceed, we take a look at the VSS changes of the three social networks (see Fig. 3) when only a posting event is performed - no other activities such as swiping and tapping is involved to remove noise. One can see from Fig. 3 that the activity transition of tweeting (from `MainActivity` to `ComposerActivity`) takes 4 time intervals[1]; the one for photo-posting in Instagram (from (`MediaCaptureActivity I` to `MediaCaptureActivity II`) takes 12 time intervals; while the one for photo-posting in Flickr (from `MainActivity` to `FilterUploadActivity`) takes

1. Here we refer a time interval as the time span needed for the retrieval of one record of the system states, which can differ from device to device but normally it is within 0.5∼0.6 seconds

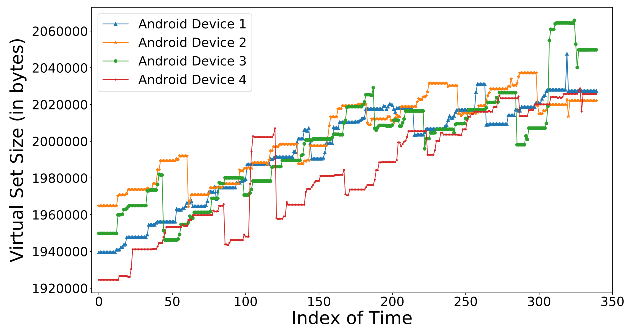

Fig. 2. The variations of *VSS* of the Twitter app in four different Android devices within approximately 2 minutes of swiping and tweeting.

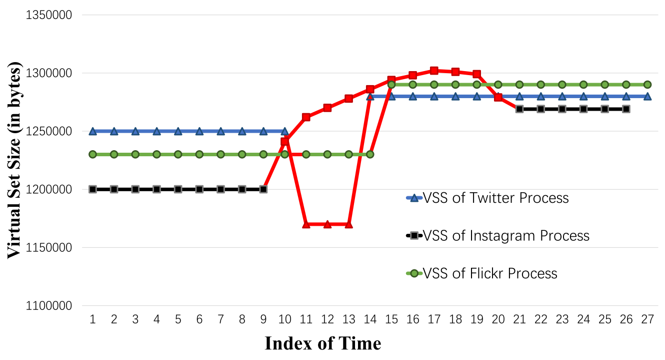

Fig. 3. The variations of *VSS* of Twitter, Instagram, and Flickr when a tweet or a photo is posted. The red parts refer to the activity transitions of tweeting/photo-posting in the corresponding social network.

only one time interval. Thus obviously, it is hard to design a uniform approach that can work for all three social networks. In this section, we propose a novel *memory pattern regression* model that can infer the activity transitions based on the VSS variations for Twitter and Instagram, and employ the RSS of the system media process[2] to infer the activity changes in Flickr. In the following, we first detail our novel memory pattern regression model.

Memory Pattern Regression. Our memory pattern regression algorithm takes as input the training set $D = \{\mathbf{y}_j\}_{j=1\cdots d}$, which is the memory data tested d times for a single activity transition with $\mathbf{y}_j = (y_{j_1}, y_{j_2}, \cdots, y_{j_{N_j}})$ being the *VSS* vector of size N_j obtained in the j-th experiment. We first need to find out the number n, which represents the number of time intervals in which a target activity transition, i.e., an activity transition of a tweeting in Twitter or a photo-posting in Instagram, takes place. In order to determine n, we first construct a sequence $\Delta\mathbf{y_j} = (\Delta y_{j_1}, \cdots, \Delta y_{j_{N_j-1}})$, where $\Delta y_{j_i} = |y_{j_{i+1}} - y_{j_i}|$; then we set up a threshold δ, which can be determined experimentally, and let n_j be the number of terms in $\Delta\mathbf{y_j}$ whose values are greater than δ; finally we set $n = \lceil\frac{\sum_{j=1}^{d} n_j}{d}\rceil$.

Having determined n, we proceed to extract the *VSS* change pattern of a transition. Instead of directly using the raw data of *VSS*, we use slopes of each pair of adjacent *VSS* points since doing this can not only mitigate the noises of memory but also obtain the shape of the transition (i.e., pattern). In order to do so, we minimize a function $f : \mathbb{R}^{n+1} \longrightarrow \mathbb{R}^{n+1}$, which is represented as

$$f \leftarrow \arg\min_{f} \left\{ \frac{1}{d}\Sigma_{j=1}^{d}[\frac{1}{n}\Sigma_{i=1}^{n}(\Delta f(j_i) - \Delta y'_{j_i})^2] \right\} \qquad (1)$$

where $\Delta f(j_i) = f(j_{i+1}) - f(j_i)$ is the slope between the point j_{i+1} and j_i since $j_{i+1} - j_i = 1$, and $\Delta y'_{j_i}$ is the ith term in the consecutive interval of $\Delta\mathbf{y_j}$ with size n that covers the maximum number of values greater than δ.

In order to learn the f satisfying (1), we employ a 5-layer feedforward neural network with n neurons at each layer. In this deep neural network, each neuron in the input and output layers is connected to a neuron (one-to-one) in the adjacent hidden layers, but all the hidden layers are fully connected. Moreover, the input and output layers simply use the Hadamard product as the activation function while all other layers use the Sinusoid function for activation.

The neural network adopted in our approach was implemented based on the Google TensorFlow [39] deep learning framework with the Adam stochastic optimization [40] that takes the right part of (1) as a loss function. By minimizing the loss function, we can obtain $\mathbf{v_j} = (\Delta f(j_1), \Delta f(j_2), \ldots, \Delta f(j_n))$ as the *VSS* pattern, which is related to an activity transition with n pieces of slopes.

Let $\mathbf{y} = (y_1, \ldots, y_N)$, with $N > n$, be the *VSS* sequence collected from a victim's device. The question here is: how can we tell whether the target activity transition happens in this sequence based on the pattern $\mathbf{v_j}$? Intuitively, directly determining whether the euclidean distance between them is smaller than a threshold σ, i.e., $\|\mathbf{v_j} - \Delta\mathbf{y}\| < \sigma$ with $\Delta\mathbf{y}$ being n consecutive slops of $\mathbf{y}$, seems reasonable. However, the euclidean distance is not applicable in our case since if a user constantly uses an app without quitting, the *VSS* of the App would accumulate. Thus clearly, statically setting a threshold can result in a gradually larger error. Therefore, we developed a comparison method which fits our problem well. Instead of directly considering $\Delta\mathbf{y}$, we tolerate a little bit by considering a neighborhood of $\Delta\mathbf{y}$, denoted by $B_r(\Delta\mathbf{y})$, where r is a small integer such that $n < r < N$. Specifically, for a *VSS* value at i, we consider the previous and post r slopes of this value, yielding $B_r(\Delta y_i) = (\Delta y_{i-r}, \ldots, y_i, \ldots, \Delta y_{i+r})$ (totally $2r+1$ terms). After that, given a threshold σ, our comparison algorithm extracts each continuous segment of slopes with size n from $B_r(\Delta\mathbf{y_j})$ and calculates the mean squared errors with $\mathbf{v_j}$, denoted as $mse_1, mse_2, \ldots, mse_{2r+2-n}$ (totally $2r+2-n$ terms). Lastly, if $\max(|mse_1 mse_2|, |mse_2 - mse_3|, \ldots, |mse_{2r+2-n} - mse_{2r+2-n}|) < \sigma$, we perceive that the target activity transition occurs in $B_r(\Delta y_i)$.

For the activity transition inference of the photo-posting event in Flickr, we employed a different approach based on the following observation: when posting a photo in Flickr, a user needs to choose a picture from his album or to take a photo; in this case, the *RSS* of the system media process, i.e., `android.process.media`, increases and the *VSS* of Flickr increases. Therefore, using this information, one can infer that an activity transition of a photo-posting event occurs in Flickr. Note that even though Instagram is also a photo-based social network, this method does not apply to it because Instagram allows a user to post comments with a

2. We employed RSS for Instagram too but our memory pattern regression model performs better.

picture. Therefore, if directly using this method in Instagram, one may not be able to tell if a user posts a photo or just sends a comment.

4.1.3 Inference of Social Network Events

To infer the timestamp of a tweeting/photo-posting event, we need to check not only the corresponding activity transitions implied by the memory state changes but also the network states of the device: the existence of an activity transition alone does not mean that the event actually has happened - it is completed only after the tweet message or the photo is sent via the network. Therefore, we need to figure out whether `tcp_snd` and `tcp_rcv` increase after the detection of the activity transition[3]. In other words, an attacker can infer when a tweeting/photo-posting event happens by combining activity transitions with the network `tcp_snd` and `tcp_rcv` information.

In Twitter, we can also infer the number of characters in a tweet message by the keyboard event. Note that after a tweet is posted, keyboard would disappear and `tcp_snd` would increase. However, we cannot use `tcp_snd` to precisely estimate the number of characters sent in a tweet due to the protocol overhead of TCP connections. In our approach, we resort to CPU usage information. We found that the system states of the Android keyboard process (`com.google.android.inputmethod.latin`) are tightly related to a user's typing actions. More specifically, when the keyboard is launched, the *VSS* of the keyboard process increases drastically from a constant state; when a user types a keystroke, the `utime` and the `stime` of the keyboard process roughly increase by 1 clock tick. Thus we took the ceiling of the averaged increased clock ticks of `utime` and `stime` to estimate the number of characters in a tweet message.

4.1.4 Associating Device With Social Network Accounts

After retrieving the targeted social network events based on the Android system states, i.e., the timestamps of the posts and the sizes of the tweets, we can repeatedly match this information with the public information of the target social network to obtain a gradually smaller list of potential social network accounts for the device. For example, from the device system states we inferred that our target victim tweeted twice in two different timestamps and estimated the sizes of these two tweets. Then we proceeded to crawl the Twitter database and filter out those who did not tweet in these two timestamps and whose tweet sizes do not match with our inferred tweet sizes. By this way we can get a reasonably small list of potential Twitter accounts that may be associated with the victim with a high probability.

4.1.5 Crawling the Social Network Events

As mentioned earlier, we need the public information of the target social network to realize the DS-SN attack. More specifically, we need to crawl the tweets/posts occurring at a specific time within a social network. However, it is not trivial to retrieve the required data from the three social networks under our consideration. We have to overcome the following challenges.

First, even though Twitter has a streaming API which provides an interface to facilitate the retrieval of the tweets for a given time period, this method does not work since the API tends to drop a large amount of unimportant data and it has a rate limit. Luckily, we found that Twitter offers a webpage called Twitter Advanced Search (TAS) via which we can retrieve a list of tweets that contain the queried keywords, locations, or user accounts, for a given time period. Since user accounts are what we want to retrieve, we cannot use accounts as a filter. Therefore, we query with letters from "a" to "z" as the filtering keywords, and the TAS server replies a list of tweets containing any letter from "a" to "z" with very minimal data loss - the Twitter server fails to return some tweets only when it thinks that these tweets are duplicates or these tweets only contain non-English contents. Hence, we leverage this method by making queries through TAS with 26 letters and set the time period to be 1 minute. We wrote a program that sends a url in the following format:

```
https://twitter.com/i/search/timeline?f=tweets&vertical=
    default&q= keyword%20since%3Astart_date%20until%3
    Aend_date&src=typd&include_available_features=1&
    include_entities=1&lang=en&max_position=TWEET-
    old_tweet_id-new_tweet_id-
    BD1UO2FFu9QAAAAAAAAETAAAAAcAAAASAAAA...A&
    reset_error_state=false
```

where *new_tweet_id* is the tweet id of the very first tweet shown in the result, while *old_tweet_id* is the id of the last tweet received. Then TAS returns a JSON file containing 20 tweets for each query sorted by the posting times. Note that this method requires us to recurrently query until we retrieve all the tweets for our desired time period. We found that in average there are approximately 30,000 - 60,000 new tweets generated globally at every minute.

Flickr offers a similar search engine that does not require the inputs of keywords, locations, and so on. In other words, by merely feeding a time period to the Flickr search engine one can retrieve all the posts of that time period. Based on our observation, Flickr follows the following format to retrieve posts:

```
https://api.flickr.com/services/rest?sort=relevance
    &parse_tags=1&content_type=7&extras=can_comment
    %2Ccount_comments%2Ccount_faves%2'Cdescription
    %2Cisfavorite%2Clicense%2Cmedia%2
    Cneeds_interstitial%2Cowner_name%2Cpath_alias%2
    Crealname%2Crotation%2Curl_c%2Curl_l%2Curl_m%2
    Curl_n%2Curl_q%2Curl_s%2Curl_sq%2Curl_t%2Curl_z
    &per_page=100&page=xx&lang=en-US&advanced=1&
    min_upload_date=start_time&max_upload_date=
    end_time&viewerNSID=xx&method=flickr.photos.
    search&'csrf=xx&api_key=xx6&format=json&hermes
    =1&'hermesClient=1&reqId=xx&nojsoncallback=1'
```

where *start_time* and *end_time* are Unix times. The server returns a JSON file containing 100 posts for each requested

3. One can use only `tcp_snd` for the event inference but our experiments indicate that both `tcp_snd` and `tcp_rcv` increase when an event occurs, which is reasonable as TCP connections involve two-way traffics.

page number. Here a page number represents the index of the Flickr post pages, with each page containing 100 posts. For example, the third page (page number three) contains the 301st to the 400th posts following a descending posting time. We wrote a program that keeps on sending the url until all pages have been returned.

In contrast, Instagram does not have any interface for returning posts based on time; instead, it provides an interface that returns posts based on locations, accounts, and hashtags. Therefore, we can only collect as many user accounts as possible for further scrutiny. Due to the nature of Instagram, almost every Instagram user follows at least one verified account (celebrity). Thus we collected all the followers of the official account of Instagram, Selena Gomez, and Taylor Swift, the top three most-followed accounts on Instagram. As a result, we obtained in total 390,592,786 users while the number of active Instagram users is about 500 million to 600 million [41]. To collect the followers of the three most-followed accounts, we wrote a program that sends a post data to the url `https://www.instagram.com/query/` with the format of

```
"q":"ig_user(xxx)
{ followed_by.after(end_cursor, step)
{count,page_info
 { end_cursor, has_next_page}, nodes {id, is_
 verified,
followed_by_viewer,    requested_by_viewer,
       full_name, profile_pic_url, username}
}
}"
```

where *end_cursor* indicates the cursor of the last retrieved user and *step* indicates the number of users returned at each round.

4.2 SN-SN Correlation Attack

Our SN-SN correlation attack is designed to assist the DS-SN attack since the latter may not always be feasible (e.g., taking too long time) due to the infrequency of successive tweets/posts in a single social network by the target victim. To understand this challenge, we carried out a measurement study over 500,008 users randomly selected from the followers of 10 celebrities in Twitter to collect the time intervals between the most recent two successive tweets. As one can see from Fig. 4, over half of these Twitter users tweeted again more than 20 days later after their last tweet. Therefore, if only exploiting the DS-SN correlation, an attacker may have to wait for a long time to collect enough data for a satisfying result. Nevertheless, a user may access different social networks with the same device at different times; thus exploiting the correlations of the accounts in different social networks may help speed up and enhance the accuracy of the device-identity association attack. In this subsection, we investigate the SN-SN correlation to identify the possible accounts in different social networks for the same user, which can help shorten the list of candidate accounts obtained from the DS-SN attack. Assume that an attacker has launched the DS-SN attack on a victim device for two social networks and obtained two lists of candidate

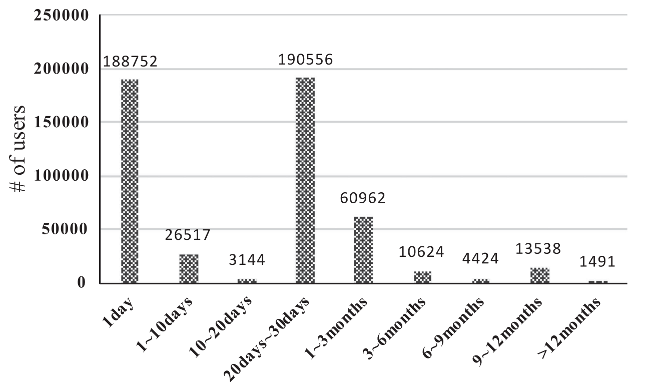

Fig. 4. The number of users *versus* the time difference between two successive tweets. One can see that the majority of the Twitter users tweeted twice between 20 days to 30 days.

accounts. Then the objective of our SN-SN attack is to identify the accounts that can match to the same user with high probability. This is done by the so-called *profile similarity*, which is a weighted average of the similarities of five attributes: *location* (s_1), *personal website link* (s_2), *username* (s_3), *biography* (s_4), and *profile image* (s_5). A novel learning-based model is presented to calculate the weight for each attribute.

In the following we present the definitions of the five attribute similarities. For better elaboration, we use $u_1^{t_1}$ and $u_2^{t_2}$ to refer to the two accounts in social network t_1 and t_2, respectively.

Definition 1 (Location Similarity, s_1). *Let $p_{1[1]}^{t_1}$ and $p_{2[1]}^{t_2}$ be the location sets by $u_1^{t_1}$ and $u_2^{t_2}$, respectively. The location similarity score s_1 is defined as*

$$s_1(p_{1[1]}^{t_1}, p_{2[1]}^{t_2}) = \begin{cases} 1 & \text{if } p_{1[1]}^{t_1} \text{ and } p_{2[1]}^{t_2} \text{ are the same,} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

In the social networks we consider, the location in a user's profile follows the format of *"City, State/Province, Country"* (a user is not mandatory to fill out all these three fields); a user can also upload the longitude and latitude of his location. In this study, we consider two locations to be the same if (1) cities are the same; or (2) states/provinces are the same; or (3) the euclidean distance between two coordinates does not exceed 814 miles, which is the approximate diameter of the largest state, Alaska, in the US [42].

Definition 2 (Personal Website Link Similarity, s_2). *Let $p_{1[2]}^{t_1}$ and $p_{2[2]}^{t_2}$ be the personal website links set by $u_1^{t_1}$ and $u_2^{t_2}$, respectively. The personal website link similarity score s_2 is defined as*

$$s_2(p_{1[2]}^{t_1}, p_{2[2]}^{t_2}) = \begin{cases} 1 & \text{if } p_{1[2]}^{t_1} = p_{2[2]}^{t_2}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Definition 3 (Username Similarity, s_3). *Let $p_{1[3]}^{t_1}$ and $p_{2[3]}^{t_2}$ be the usernames of $u_1^{t_1}$ and $u_2^{t_2}$, respectively. The username similarity score s_3 is defined as*

$$s_3(p_{1[3]}^{t_1}, p_{2[3]}^{t_2}) = \frac{|lcs(p_{1[3]}^{t_1}, p_{2[3]}^{t_2})|}{\max(|p_{1[3]}^{t_1}|, |p_{2[3]}^{t_2}|)} \tag{4}$$

where $lcs(str_1, str_2)$ denotes the longest common substring *of strings str_1 and str_2.*

According to the latent semantic analysis (LSA) in NLP [43], we defined the biography similarity in the following manner.

Definition 4 (Biography Similarity, s_4). *Let $p_{1[4]}^{t_1}$ and $p_{2[4]}^{t_2}$ be the biographies of $u_1^{t_1}$ and $u_2^{t_2}$, respectively, where $p_{1[4]}^{t_1}$ contains m_1 unique words and $p_{2[4]}^{t_2}$ contains m_2 unique words. The frequency vector $\overrightarrow{fq}(p_{1[4]}^{t_1,4}, p_{2[4]}^{t_2})$ is defined as the number of times each word of $p_{1[4]}^{t_1}$ appears in $p_{2[4]}^{t_2}$, i.e.,*

$$\overrightarrow{fq}(p_{1[4]}^{t_1}, p_{2[4]}^{t_2}) = (z_1, z_2, \cdots, z_{m_1})^T$$

with z_i being the number of times the ith word of $p_{1[4]}^{t_1}$ appearing in $p_{2[4]}^{t_2}$. Let $F = \overrightarrow{fq}(p_{1[4]}^{t_1}, p_{2[4]}^{t_2}) \cdot \overrightarrow{fq}(p_{1[4]}^{t_1}, p_{2[4]}^{t_2})^T$. We perform the SVD decomposition against F and retrieve F's singular value vector, which is denoted by $\delta = (\delta_1, \ldots, \delta_{m_1})$. Then the biography similarity $s_4(p_{1[4]}^{t_1}, p_{2[4]}^{t_2})$ is defined as

$$\begin{aligned} s_4(p_{1[4]}^{t_1}, p_{2[4]}^{t_2}) &= \min\left(\frac{\|\Sigma\|_F}{\left|p_{1[4]}^{t_1}\right|\left|p_{2[4]}^{t_2}\right|}, 1\right) \\ &= \min\left(\frac{\sqrt{\sum_{i=1}^{m} \delta_{m_1}^2}}{\sqrt{m_1 \cdot m_2}}, 1\right) \end{aligned} \tag{5}$$

Lastly, due to the special nature of profile images, we consider a profile image with and without human faces separately.

Definition 5 (Profile Image Similarity, s_5). *Let $p_{1[5]}^{t_1}$ and $p_{2[5]}^{t_2}$ be the profile images of $u_1^{t_1}$ and $u_2^{t_2}$, respectively. The profile image similarity score s_5 is defined as:*

$$s_5(p_{1[5]}^{t_1}, p_{2[5]}^{t_2}) = \begin{cases} s_5^F(p_{1[5]}^{t_1}, p_{2[5]}^{t_2}), \text{both have faces;} \\ s_5^{\neg F}(p_{1[5]}^{t_1}, p_{2[5]}^{t_2}), \text{none has face;} \\ 0, \quad\quad \text{otherwise.} \end{cases} \tag{6}$$

For the case when both profile images contain human faces, our approach compares the facial similarity in three steps: (1) *face detection*, (2) *face aligment*, and (3) *face similarity comparison*. In the face detection step, we leverage the Haar Feature-based Cascade Classifier proposed by [44][45]; in the face alignment step, we leverage the Multitask Cascaded Convolutional Network proposed and implemented by Zhang *et al.* [46]; and finally, we come up with the face similarity definition motivated by the FaceNet proposed by Schroff *et al.* [47]. Embedding [47] is a function f_{emb} whose L^2 norm satisfies $\|f_{emb}(x)\|_2 = 1$. In our study, we define f_{emb} as

$$f_{emb}(px_{i,j}) = \frac{px_{i,j}}{\sqrt{\sum_i \sum_j px_{i,j}^2}}$$

where $px_{i,j}$ is the pixel on the i-th row and j-th colum of an aligned face picture matrix. We define the loss function for the profile images with faces as

$$loss_f = \sum_{i,j}\left[\left\| f_{emb}(px_{i,j}^{p_1^{t_1}}) - f_{emb}(px_{i,j}^{p_2^{t_2}})\right\|_2^2\right] \tag{7}$$

Definition 5.1 Profile Image Similarity with Face, s_5^F. *The face image similarity is defined as*

$$s_5^F(p_{1[5]}^{t_1}, p_{2[5]}^{t_2}) = \max(1 - \sqrt{loss_f}, 0) \tag{8}$$

For the case when both images do not have human faces, we leverage the traditional algorithm, the scale invariant feature transform (SIFT) [48], used for object recognition in computer vision. Let $g(x)$ represent the number of SIFT features of picture x, and $m(x, y)$ be the number of matched features between picture x and y with Lowes's ratio test [49].

Definition 5.2 Profile Image Similarity without Face, $s_5^{\neg F}$.

$$s_5^{\neg F}(p_{1[5]}^{t_1}, p_{2[5]}^{t_2}) = \frac{m(p_{1[5]}^{t_1}, p_{2[5]}^{t_2})}{\min(g(p_{1[5]}^{t_1}), g(p_{2[5]}^{t_2}))} \tag{9}$$

Now we are ready to define our overall profile similarity:

Definition 6 (Profile Similarity). *Let $u_1^{t_1} \in SN_1$ and $u_2^{t_2} \in SN_2$, the profile similarity $S : SN_1 \times SN_2 \to [0, 1]$ of two user accounts is defined as*

$$S(u_1^{t_1}, u_2^{t_2}) = \sum_{k=1}^{5} w_k s_k(p_{1[k]}^{t_1}, p_{2[k]}^{t_2}) \tag{10}$$

where $s_k(p_{1[k]}^{t_1}, p_{2[k]}^{t_2})$ represents the k-th attribute similarity of the two user accounts and w_k is the corresponding weight that captures the impact of s_k on the overall profile similarity.

In the following we present a learning based model to calculate the weights in Eq. (10). Intuitively, the weights should be determined by maximizing the probability of correct matches and minimizing the probability of incorrect matches. Suppose we have a ground truth dataset which includes n_{gt} users from SN_1 and n_{gt} users from SN_2, with the ith user from SN_1 and the ith user from SN_2 are indeed the same person. Based on the profile similarity defined in Eq. (10), we can get a matrix S over the pairwise profile similarity scores between SN_1 and SN_2.

$$\mathbf{S}(SN_1, SN_2) = \begin{pmatrix} S(u_1^1, u_2^1) & S(u_1^1, u_2^2) & \ldots & S(u_1^1, u_2^{n_{gt}}) \\ S(u_1^2, u_2^1) & S(u_1^2, u_2^2) & \ldots & S(u_1^2, u_2^{n_{gt}}) \\ \vdots & \vdots & \ddots & \vdots \\ S(u_1^{n_{gt}}, u_2^1) & S(u_1^{n_{gt}}, u_2^2) & \ldots & S(u_1^{n_{gt}}, u_2^{n_{gt}}) \end{pmatrix}$$

where each entry $S(u_1^i, u_2^j)$ is the profile similarity score defined in Eq. (10).

In order to find the proper weights, instead of considering only the incorrect matches or correct matches, we came up with a novel loss function which takes into consideration both the correct matches and the incorrect matches, making it more efficient and effective. The loss function is defined as follows:

$$loss = l(w_1, w_2, \ldots, w_k) =$$

$$\left\langle \left[\begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}_{n_{gt}\times 1} - diag(\boldsymbol{S}) + \frac{1}{n_{gt}-1} \begin{pmatrix} \sum_{j,j\neq 1} S(u_1^1, u_2^j) \\ \sum_{j,j\neq 2} S(u_1^2, u_2^j) \\ \vdots \\ \sum_{j,j\neq n_{gt}} S(u_1^{n_{gt}}, u_2^j) \end{pmatrix} \right]^2 \right\rangle \quad (11)$$

where $\langle\cdot\rangle$ represents the expectation sign. The weight vector is determined by solving the following system of equations.

$$\nabla l = 0 \Longrightarrow \begin{cases} \frac{\partial l}{\partial w_1} = 0 \\ \frac{\partial l}{\partial w_2} = 0 \\ \vdots \\ \frac{\partial l}{\partial w_k} = 0 \end{cases} \quad (12)$$

In our approach, we solve the system of equations (12) by TensorFlow with the gradient descent optimization [50].

5 Evaluation

In this section, we report the evaluation results of our approach to demonstrate its effectiveness on the DS-SN attack within a single social network and the SN-SN attack across two social networks.

According to the design elaborated in Section 4, we implemented the following programs: a malicious app in Java with 3280 lines of code (LOC) for collecting the publicly available Android system state data, which deals with the first step of the DS-SN attack; a server code for activity transition inference and social network event inference with 1800 LOC in Python, which handles the second and third steps of the DS-SN attack; a program of 1595 LOC in Python to crawl the databases of Twitter, Instagram, and Flickr, and perform the device-account matching, which corresponds to the fourth step of the DS-SN attack; and finally, a program with 707 LOC in Python, which performs the SN-SN user account association analysis.

In our experiments, we used five different Android devices: a Nexus 7 with Android version 6.0.1, a HTC One Sense 6 with Android version 5.0.2, a Blu R1 HD with Android version 6.0, a Huawei Honor 8 Lite with Android version 7.0, and a Samsung Galaxy S4 with Android version 4.2.2. In the server side, we used a Dell Inspiron 5559 Specs with Intel Core i7-6500U CPU @ 2.50GHz x 4 running OS Ubuntu 16.04 LTS to conduct the experiments.

5.1 Effectiveness of the DS-SN Correlation Attack

We evaluated the effectiveness of our DS-SN attack by i) validating the memory regression learning model, ii) inferring the posting timestamps, iii) inferring the number of characters in a tweet, and iv) inferring the possible list of social network accounts associated with the victim device via multiple posting events.

5.1.1 Performance of the Memory Regression Model

As mentioned in Section 4, we leveraged a 5-layer feedforward neural network to learn the patterns of activity transitions in Twitter and Instagram, i.e., from `MainActivity` to `ComposerActivity` in Twitter and from `MediaCaptureActivity I` to `MediaCaptureActivity II` in Instagram. We wrote an automatic program in Python using the Android debug bridge (`adb`) [51] with the UI Automator framework for the generation and collection of the training data and test data. We ran the automatic `adb` Python program in all five Android devices for collecting the training data. As a result, for each of the 5 Android devices, we collected 1,900 pieces of training data for Twitter and 2,340 pieces of training data for Instagram. Thus, in total we have 9,500 pieces of training data for Twitter and 11,700 pieces of training data for Instagram. Note that each piece of the training data contains only the target activity transition - no other noisy actions such as swiping. To determine n as mentioned in Section 4, we tried multiple different values for the threshold δ and found that the values around 1000 work well for both Twitter and Instagram; thus we set $\delta = 1000$. Then we calculated the values of n for Twitter and Instagram, and obtained 7 and 15, respectively. Also, we set the cessation threshold for the mean squared error to be 10^{-6}. As a result, the mean squared error for the learning process of the Twitter activity transition drops to below 10^{-6} after approximately 30 minutes and the one for the Instagram activity transition drops to below 10^{-6} after approximately 40 minutes, which are reasonably low for deep learning models.

5.1.2 Inference of the Posting Timestamps

The automatic `adb` Python program performed 5,625 tests for all five Android devices, with each device performing 1,125 tests. Each test is for one target social network app and collects a block of data with 6 rows: *VSS*, `tcp_snd`, and `tcp_rcv` for the social network app, `utime` and `stime` for the keyboard process, and *RSS* for the media process, and each row contains 600 values. Therefore, for each Android device, we conducted 375 tests for each target app. It took 5 to 6 minutes to perform one test, during which our automatic `adb` program controlled the target social network app to tweet with a random number of characters (Twitter) or to post a photo (Instagram and Flickr)), and recorded the system timestamps of the tweet/photo-post event (ground truth timestamps) for evaluation. The `adb` program also generated random noisy actions such as swiping, tapping, and commenting in the target apps to imitate human beings' actions on these apps during each block collection. Meanwhile, our malicious app collected *VSS*, `tcp_snd`, and `tcp_snd` for the three social network apps and the *RSS* of Android media process as shown in Table 2 for the inference attack.

We considered that a correct inference is made if the difference between the ground truth posting timestamp and our inferred posting timestamp is less than 30 seconds. Table 3 reports the averaged inference results of all Android devices. As one can see, the numbers of correct inferences of the posting timestamps for Twitter, Flickr, and Instagram are 86.99%, 96.81%, and 96.73%, respectively. Table 4 reports the inference results of each Android device against the three target apps. One can see that our approach performs the best for Flickr, with Instagram the second and

TABLE 3
Averaged Inference Results on the Posting Timestamps and the Number of Characters in Tweets

App Name	Averaged Inference Results on the Posting Timestamps and the Number of Characters in Tweets			
	Types of Random Noisy Actions Performed	Accuracy of Inferrring Posting Timestamps	Average Difference of Timestamps (in seconds)	Accuracy of Inferring Number of Characters in Tweets
Twitter	1. Swiping 2. Tapping 3. Commenting	86.99%	16.35	37.18% ($\epsilon = 0.05$) 91.86% ($\epsilon = 0.15$) 98.92% ($\epsilon = 0.3$)
Flickr	1. Swiping 2. Tapping 3. Commenting	96.81%	15.05	N/A
Instagram	1. Swiping 2. Tapping 3. Commenting	96.73%	19.75	N/A

Twitter the third. We probed into it and found that this phenomenon may be due to the following reason: Flickr has clear activity transitions causing distinct changes on side-channels while Twitter has relatively mild activity transitions with less distinct changes on side-channels. Considering the inference success rates of different devices, one can see that Nexus 7 performs the best, followed by Blu R1 HD, Huawei Honor A, Blu R1 HD, Samsung Galaxy S4, and HTC One Sense 6, in descending order. This might be because devices having better inference performances are mounted with more advanced CPUs and/or installed with newer versions of the Android, which can help optimize the processing and rendering of images, resulting in more precise inference results. Note that in our memory regression model we set $\sigma = 10^5$, which was determined based on many trials.

5.1.3 Inference of the Number of Characters in a Tweet

To evaluate the inference accuracy on the number of characters in a tweet through CPU `utime` and `stime` of the Keyboard process, we denote the actual number of characters of a tweet by N_{act}, the inferred number by N_{inf}, and set an error parameter ϵ such that our inference is considered a success if the error percentage is less than ϵ, i.e., $\frac{|N_{act}-N_{inf}|}{N_{act}} \leq \epsilon$. In order to evaluate the accuracy of our inference attacks, we randomly sampled 5,000 tweets from those of the 500,008 Twitter users collected in Section 4, and re-tweeted these 5,000 tweets using our Android devices. By doing so, we can guarantee that the distribution of the number of characters of our posted tweets follows the one in real-world. The average number of characters of these 5,000 tweets was 189. The results are reported in Table 3, which indicate that if we set ϵ to be 0.05, the possibility of correct inference is 37.18%; if ϵ is set to 0.15, the possibility of correct inference is 91.86%; while if ϵ is set to 0.3, the possibility increases to 98.92%.

5.1.4 Associating Device With Social Network Accounts

We used Twitter to illustrate the performance of associating the victim device with a list of candidate social network accounts. In this evaluation, we conducted two studies, with one on the Twitter celebrities and one on five volunteers. Note that the study on Twitter celebrities cannot identify any device that is associated with a social network account because we do not install our malicious app in any legitimate user in Twitter; nevertheless, the tweeting activities of celebrities can facilitate us to figure out how many people simultaneously tweet at two or more timestamps, which provides a perfect justification to explain why we can correlate the tweeting activities at different timestamps to identify the user account associated with a victim device.

For the study on celebrities, we collected tweets from 5 active Twitter celebrities, with 10 tweets for each at different timestamps, from November 2016 to February 2017, to get the data at 50 different timestamps. Each celebrity has at least 20 million followers. We considered celebrities as our study subjects because they not only use real identities, but also are active on Twitter. The results are reported in Table 5, which indicate that the average number of Twitter users who tweeted at one timestamp is 52,016, at the same two timestamps is 359, at the same three timestamps is 61, and at the same four timestamps is 6. These results are interesting as one can see that the number of users who tweeted simultaneously at multiple timestamps drops significantly as the number of considered timestamps increases.

Besides using celebrities as testing subjects mentioned above, we also recruited five volunteers to participate in our

TABLE 4
Inference Results on the Posting Timestamps for Each Device

Device Name	App Name			
	Twitter	Instagram	Flickr	Average
Nexus 7	90.67%	98.67%	99.73%	96.36%
HTC One Sense 6	82.93%	91.20%	90.40%	88.18%
Blu R1 HD	91.47%	87.47%	98.67%	92.54%
Huawei Honor 8	84.27%	94.93%	98.40%	92.53%
Samsung Galaxy S4	90.93%	85.87%	98.93%	91.91%
Average	88.05%	91.63%	97.23%	N/A

TABLE 5
Attack Results on Celebrities

Attack Results on Celebrities	
# of Timestamps	Average Number of Potential Twitter Identities
1	52,016
2	359
3	61
4	6

TABLE 6
Attack Results on Volunteers

Attack Results on Volunteers												
# of Timestamps	Number of Potential Twitter Identities (Without Number-of-Character Filter)						Number of Potential Twitter Identities (With Number-of-Character Filter)					
	Participant 1	Participant 2	Participant 3	Participant 4	Participant 5	Average	Participant 1	Participant 2	Participant 3	Participant 4	Participant 5	Average
1	52,120	61,810	55,132	51,205	50,037	54,060	2,313 ($\epsilon = 0.3$)	2,538 ($\epsilon = 0.3$)	2,272 ($\epsilon = 0.3$)	1,971 ($\epsilon = 0.3$)	2,067 ($\epsilon = 0.3$)	2,232 ($\epsilon = 0.3$)
2	369	403	331	324	347	354	21 ($\epsilon = 0.3$)	19 ($\epsilon = 0.3$)	22 ($\epsilon = 0.3$)	12 ($\epsilon = 0.3$)	10 ($\epsilon = 0.3$)	16 ($\epsilon = 0.3$)
3	52	61	71	43	57	56	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	2 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)
4	11	8	13	9	7	9	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)
5	4	3	5	4	3	3	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)
6	3	2	2	1	1	1	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$))	1 ($\epsilon = 0.3$)
7	3	2	1	1	1	1	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)
8	1	2	1	1	1	1	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)
9	1	1	1	1	1	1	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)	1 ($\epsilon = 0.3$)

evaluation process. Each participant used one of our five Android devices to tweet 10 times at 10 different timestamps with his/her own Twitter account. Each tweet contained a random number of characters. The detailed results are shown in Table 6. If we do not apply the number of tweeted characters as a filter, we found that in average there are 54,060 users who tweeted at one timestamp as the volunteers did. Then the number goes to 354 at two same timestamps, 56 at three same timestamps, 9 at four same timestamps, 3 at five same timestamps, and 1 at six same timestamps. These numbers indicate that an attacker needs to conduct the inference attack for an average of six times in order to associate the Twitter account of one of our volunteers with his/her device without applying the number of characters as a filter. If we apply this filter, as one can see from the results, an attacker only needs to conduct the inference attack for an average of three times to successfully associate one volunteer with his/her Twitter account.

In order to assess the influences of human behavioral factors, we asked each of the five participants to use all the five Android devices shown in Table 4 to send posts in all three social networks. More specifically, for each device and each social network, each participant sent 10 posts. Therefore, we obtained in total 750 ($5 \times 5 \times 10 \times 3$) posts. We applied our approach and found that the inference success rates are close to those shown in Table 4, with the highest deviation less than 4%. Thus one can conclude that human factors do not have a significant impact on our approach.

5.2 Performance of the SN-SN Correlation Attack

As one can see from the results shown above, an attacker needs to infer at least three timestamps on average to correlate a user's social network account with his/her device with the estimated number of characters as a filer. Without applying this filter, on average at least 6 timestamps need to be inferred. To speed up this process, SN-SN correlation attack can be launched. In this subsection, we evaluate the performance of the SN-SN correlation attack.

Since Instagram does not provide an interface via which we can retrieve posts at any given timestamp, we considered the other two social networks: Twitter and Flickr. To proceed, we need to first derive the weight for each attribute in order to calculate the profile similarity. For this purpose we need ground truth data to train our learning model proposed in Section 4.2. We manually checked the tweets of a large amount of Twitter users and filtered out those who do not explicitly provide their Flickr links in one of their tweets. Finally, we obtained 20 pairs of Twitter and Flickr users we were certain that each pair actually refers to the same person. Then we applied these data to minimize the loss function in equation (11). Finally, we obtained the following weights when the loss function reaches its stationary minimum of 0.108886: $\omega_1 = 0.509717$, $\omega_2 = 0.194603$, $\omega_3 = 0.51427$, $\omega_4 = 0.171117$ and $\omega_5 = 0.30354$. Lastly, we normalized these weights by setting $\omega_i^* = \frac{\omega_i}{|\sum_{i=1}^{5} \omega_i|}$, and obtained

$$\omega_1^* = 0.301029,\ \omega_2^* = 0.114929,\ \omega_3^* = 0.303718,\ \omega_4^* = 0.101058,$$

and $\omega_5^* = 0.179265$.

As one can see from the above, username (ω_3) plays an extremely important role in determining whether two users from two different social networks are actually the same person, followed by location (ω_1). The least important factor is biography (ω_4), which indicates that people tend to write differently for different social networks.

Next we analyzed the Twitter-Flickr matching performance. We asked each of the five volunteers to send a post in Flickr with his/her Flickr account after he/she finished the 10 tweeting experiments mentioned above. Then we correlated Flickr with Twitter by considering only the first timestamp of the Twitter data of all the participants. Next we crawled all the posts in Flickr at the five timestamps corresponding to those at which our five participants sent posts. As a result, we found that there were totally 611 Flickr users who also posted at the same timestamp at which the first participant posted. The numbers of Flickr users who posted at the same time as the second, third, fourth, and fifth participant were 533, 421, 498, and 506, respectively. Therefore, for the first participant, we analyzed a similarity matrix S_{p_1} defined in Section 4 with 1,413,243 entries ($2,313 \times 611$). Similarly, the numbers of entries in the similarity matrices for the second, third, fourth, and fifth participant were 1,352,754 ($2,538 \times 533$), 956,521 ($2,272 \times 421$), 945,204 ($1,898 \times 498$), and 1,112,188 ($2,198 \times 506$), respectively. Ideally, the majority of the entries in these five matrices should contain very small values, i.e., close to zero; while the entries for the correct matches contain big values, i.e., close to one.

After we evaluated the five matrices with the trained weights obtained above, the similarity matrix of the first participant shows that 1,332,575 entries (94.29%) are below 0.1; only 3 pairs whose similarity scores are greater than 0.5; and only 1 pair whose similarity score is greater than 0.65, which is the correct matching, i.e., the Twitter account and

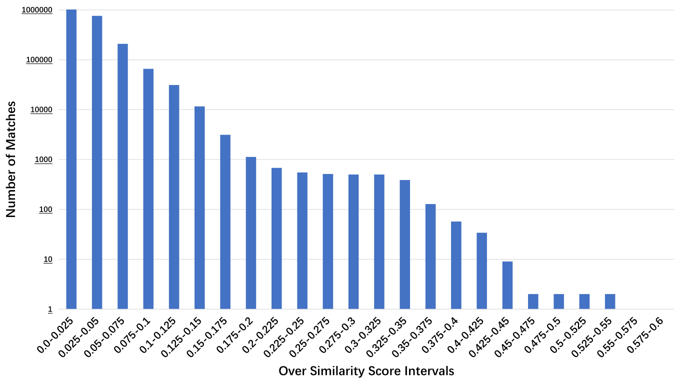

Fig. 5. The distribution of the matches of the fifth participant. The x-axis is the intervals of overall similarity scores while the y-axis is the number of matches.

the Flickr account of the first participant. Similarly, our results showed that over 90% of all the entries of the similarity matrices of the other four participants are below 0.1; and the correct matches corresponds to the pairs with the highest scores. Among the results of all the five participants, the highest overall similarity score is the one for the fifth participant, which is 0.84. Fig. 5 illustrates the detailed distribution of the matches of the fifth participant as an example. These results not only indicate that both our definitions of the attribute similarities and the weights work well in real cases, but also imply that our SN-SN correlation attack is effective in facilitating the DS-SN attack to speed up the device-identity association attack.

6 Conclusions and Future Research

In this paper, we proposed a novel attack architecture to associate a mobile device user with his social network accounts. Our architecture is composed of two attack vectors: DS-SN correlation attack and SN-SN correlation attack. We used five Android devices and three social networks, Twitter, Flickr, and Instagram, to demonstrate and evaluate our architecture. In the DS-SN attack, we inferred a user's tweeting/photo-posting timestamps and tweet sizes based on the device's publicly available system states, i.e., CPU, memory, and network data; then we matched the timestamps and tweet sizes with the social network events to uncover the user's accounts. When it is hard to collect sufficient data for DS-SN attacks as the victim may not tweet/post very frequently, SN-SN attack can be employed to investigate the correlations of different social network accounts belonging to the same person. SN-SN attack compares the profile similarity of two accounts and finds out the best match. Our experimental results over the three popular social networks corroborate the effectiveness of our attack architecture.

Note that the approach proposed in this paper cannot effectively associate multiple different mobile devices used by the same user to access different social networks, i.e., the so-called DS-DS attack. In order to realize effective DS-DS attacks, more side channel information such as calibration data [21] and Wi-Fi network traces [10] might be needed. This is a complicated research topic with strong security significance and will be investigated in our future research.

Acknowledgments

The authors would like to thank all funding agencies.

References

[1] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," in *Proc. 30th IEEE Symp. Secur. Privacy*, 2009, pp. 173–187.
[2] M. Srivatsa and M. Hicks, "Deanonymizing mobility traces: Using social network as a side-channel," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 628–637.
[3] S. Nilizadeh, A. Kapadia, and Y.-Y. Ahn, "Community-enhanced de-anonymization of online social networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 537–548.
[4] H. Li, C. Zhang, Y. He, X. Cheng, Y. Liu, and L. Sun, "An enhanced structure-based de-anonymization of online social networks," in *Proc. Int. Conf. Wireless Algorithms Syst. Appl.*, 2016, pp. 331–342.
[5] Why has a data leak of 1 billion social media profiles occurred?, 2020. [Online]. Available: http://www.digitaljournal.com/tech-and-science/technology/why-has-a-data-leak-of-1-billion-social-media-profiles-occurred/article/562302
[6] Plugging the leak: Data loss and smartphones, 2020. [Online]. Available: https://blog.shi.com/solutions/plugging-the-leak-data-loss-and-smartphones/
[7] X. Zhou *et al.*, "Identity, location, disease and more: Inferring your secrets from android public resources," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 1017–1028.
[8] Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers, 2018. [Online]. Available: https://www.frida.re/
[9] J. Choi, H. Cho, and J. Yi, "Personal information leaks with automatic login in mobile social network services," *Entropy*, vol. 17, pp. 3947–3962, 06 2015.
[10] L. Yu, B. Luo, J. Ma, Z. Zhou, and Q. Liu, "You are what you broadcast: Identification of mobile and IoT devices from (public) WiFi," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 55–72. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/yu
[11] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, and W. Lv, "Edge computing security: State of the art and challenges," *Proc. IEEE*, vol. 107, no. 8, pp. 1608–1631, 2019.
[12] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into your app without actually seeing it: UI state inference and novel android attacks," in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 1037–1052.
[13] M. Li *et al.*, "When CSI meets public WiFi: Inferring your mobile phone password via WiFi signals," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1068–1079.
[14] Q. Yang, P. Gasti, G. Zhou, A. Farajidavar, and K. Balagani, "On inferring browsing activity on smartphones via USB power analysis side-channel," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 5, pp. 1056–1066, May 2017.
[15] C. Song, F. Lin, Z. Ba, K. Ren, C. Zhou, and W. Xu, "My smartphone knows what you print: Exploring smartphone-based side-channel attacks against 3D printers," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 895–907.
[16] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 368–379.
[17] V. van der Veen *et al.*, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1675–1689.
[18] H. Li, Y. He, L. Sun, X. Cheng, and J. Yu, "Side-channel information leakage of encrypted video stream in video surveillance systems," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[19] G. Yang, J. Huang, G. Gu, and A. Mendoza, "Study and mitigation of origin stripping vulnerabilities in hybrid-postmessage enabled mobile applications," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 742–755.
[20] X. Zhang *et al.*, "An empirical study of web resource manipulation in real-world mobile applications," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1183–1198.
[21] J. Zhang, A. R. Beresford, and I. Sheret, "SensorID: Sensor calibration fingerprinting for smartphones," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 638–655.

[22] T. Brennan, N. Rosner, and T. Bultan, "JIT leaks: Inducing timing side channels through just-in-time compilation," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 1207–1222. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00007

[23] M. Backes, M. Humbert, J. Pang, and Y. Zhang, "walk2friends: Inferring social links from mobility profiles," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1943–1957.

[24] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A practical attack to de-anonymize social network users," in *Proc. IEEE Symp. Secur. Privacy*, 2010, pp. 223–238.

[25] S. Ji, W. Li, N. Z. Gong, P. Mittal, and R. A. Beyah, "On your social network de-anonymizablity: Quantification and large scale evaluation with seed knowledge," in *Proc. 22nd Annu. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–15.

[26] S. Lai, H. Li, H. Zhu, and N. Ruan, "De-anonymizing social networks: Using user interest as a side-channel," in *Proc. IEEE/CIC Int. Conf. Commun. China*, 2015, pp. 1–5.

[27] A. Chaabane, G. Acs, M. A. Kaafar, "You are what you like! information leakage through users' interests," in *Proc. 19th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2012, pp. 1–14.

[28] B. Mei, Y. Xiao, R. Li, H. Li, X. Cheng, and Y. Sun, "Image and attribute based convolutional neural network inference attacks in social networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 2, pp. 869–879, Apr.–Jun. 2020.

[29] N. Z. Gong and B. Liu, "You are who you know and how you behave: Attribute inference attacks via users' social friends and behaviors," in *Proc. 25th USENIX Secur. Symp.*, USENIX Assoc., Aug. 2016, pp. 979–995. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/gong

[30] W. U. Hassan, S. Hussain, and A. Bates, "Analysis of privacy protections in fitness tracking social networks-or-you can run, but can you hide?," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 497–512.

[31] M. Mondal *et al.*, "Moving beyond set-it-and-forget-it privacy settings on social media," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 991–1008.

[32] C. Zhang, H. Jiang, Y. Wang, Q. Hu, J. Yu, and X. Cheng, "User identity de-anonymization based on attributes," in *Wireless Algorithms, Systems, Appl.*, E. S. Biagioni, Y. Zheng, and S. Cheng, Eds., Berlin, Germany: Springer, 2019, pp. 458–469.

[33] C. Zhang, S. Wu, H. Jiang, Y. Wang, J. Yu, and X. Cheng, "Attribute-enhanced de-anonymization of online social networks," in *Comput. Data Social Netw.*, A. Tagarelli and H. Tong, Eds., Berlin, Germany: Springer, 2019, pp. 256–267.

[34] Massive android malware outbreak invades Google play store, 2017. [Online]. Available: http://fortune.com/2017/09/14/google-play-android-malware/

[35] Cve-2017–0561 detail, 2017. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2017–0561

[36] Overview of android memory management, 2016. [Online]. Available: https://developer.android.com/topic/performance/memory-overview.html

[37] Debug.memoryinfo, 2016. [Online]. Available: https://developer.android.com/reference/android/os/Debug.MemoryInfo.html

[38] Android toolbox, 2016. [Online]. Available: http://elinux.org/Android_toolbox

[39] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.

[40] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2015, *arXiv:1412.6980*.

[41] Instagram: Number of monthly active users 2013–2016, 2016. [Online]. Available: https://www.statista.com/statistics/253577/number-of-monthly-active-instagram-users/

[42] Alaska, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Alaska

[43] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse Processes*, vol. 25, no. 2/3, pp. 259–284, 1998.

[44] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2001, p. I.

[45] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Proc. Int. Conf. Image Process.*, 2002, p. I.

[46] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Process. Lett.*, vol. 23, no. 10, pp. 1499–1503, Oct. 2016.

[47] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 815–823.

[48] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, 1999, pp. 1150–1157.

[49] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[50] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.

[51] Android debug bridge, 2017. [Online]. Available: https://developer.android.com/studio/command-line/adb.html