Beyond the Layers: Identifying Novel Privacy Threats for Internet Users

レイヤーを越えて: インターネットユーザに対する未知なるプライバシー脅威の特定

February, 2020

Takuya Watanabe

渡邉 卓弥

Beyond the Layers: Identifying Novel Privacy Threats for Internet Users

レイヤーを越えて: インターネットユーザに対する未知なるプライバシー脅威の特定

February, 2020

Waseda University

Graduate School of Fundamental Science and Engineering

Department of Computer Science and Communications Engineering,
Research on Networked Systems

Takuya Watanabe

渡邉 卓弥

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

With the spread of smartphones and internet of things (IoT), the number of devices connected to the Internet reached 22 billion at the end of 2018 [1]. Modern Internet devices with rich features store privacy data including user identity, activity logs, location, video, and audio. This means more opportunities for users' data to be exposed. In light of these increasing privacy risks, efforts to protect privacy have become a global trend. Previous research on capturing and analyzing attacker behavior [2–8] has resulted in beneficial countermeasures to protect user privacy. In addition, laws that strongly restrict privacy access, such as General Data Protection Regulation (GDPR) [9] and California Consumer Privacy Act (CCPA) [10], have been enforced in many countries.

Unfortunately, privacy threats to Internet users are intensifying daily. Typical attacks by which an attacker exploits application security holes, or vulnerabilities, remain active. In 2018, Facebook, the largest social network, revealed that information of 50 million users was leaked due to a bug in which one user was given another user's access token [11]. In addition, the latest attackers are no longer limited to behavior of computer systems. A side-channel indirectly estimates confidential information by measuring the side-effects of computer systems on physical space. Horn et al. discovered Meltdown [12] and Spectre [13], which cause timing attacks

on CPUs, and showed that most CPUs released in the last 25 years have a potential risk of data leakage. Phishing attacks exploit human psychological weaknesses such as fear and curiosity. In just three months from April to June 2019, Kaspersky [14] detected 129.9 million phishing attacks.

As these examples show, attackers attempt to exploit information outside computer system, i.e., *beyond the layers*. Such attacks and defenses across layers are new areas of research that should be highlighted to ensure user privacy. To protect users from modern cyber attacks, it is necessary to understand the interaction between applications, physical space, networks, and human factors. This thesis focuses on the following new threat scenarios that exploit resources outside computer systems.

Exploiting Side Channel on Physical World. The latest mobile and IoT devices are equipped with sensors to provide users with rich experiences linked to the real world. These sensors collect data such as acceleration and magnetic fields on the device. These data reflect the physical behavior of the device's user and may leak his/her privacy. Access to physical sensors on most devices are not protected by permission mechanisms.

Exploiting Side Channel on Network. With higher quality communication lines, network noise no longer has a significant effect on communication speed. The communication time mainly varies depending on the state of the server side. In other words, an attacker can estimate the state of a remote application by observing the communication time. By combining several application states, it may be possible to estimate more specific user data. Countermeasures that conceal communication time are not common.

Exploiting Human Perception. Users decide what changes to make to computer systems based on their expectations. Metadata, such as text description describing the behavior of an application, are useful for users to decide whether to install an application. The user can be misled by the text description to install an application that accesses sensitive data against the user's expectation. There is no guideline in app markets to ensure that the correct description is written.

To measure the impact of attacks beyond the layers, we adopt both a conceptual approach that demonstrates potential threats by building novel attacks (Chapter 2, 3)

Table 1.1 Summary of the layers that each chapter focuses on.

	Approach	Layer	Exploited Channel
Chapter 2	conceptual	physical	user activity in the real world captured by motion sensors
Chapter 3	conceptual	network	time required for communication and server processing
Chapter 4	empirical	human	text description that shows behavior of applications

and an empirical approach that reveals how pervasive the threat is through large-scale analysis in the wild (Chapter 4).

1.2 Thesis Contributions

The goal of this thesis is to identify novel privacy threats for Internet users that exploit information beyond the layers. We explore three attack surfaces on the lower and upper layers that impact major Internet usages of modern consumers. The layers that each chapter focuses on are summarized in Table 1.1. This thesis is organized as follows:

Sensor-based user location tracking is presented in Chapter 2. In this chapter, we demonstrate the sensor data that collects the user's behavior leaks real-world location information of the user. We developed proof-of-concept framework called *RouteDetector*, which identifies a route for a train trip by simply reading smart device sensors: an accelerometer, magnetometer, and gyroscope. All these sensors are commonly used by many apps without requiring any permissions. The key technical components of RouteDetector can be summarized as follows. First, by applying a machine-learning technique to the data collected from sensors, RouteDetector detects the activity of a user, i.e., "walking," "in moving vehicle," or "other." Next, it extracts departure/arrival times of vehicles from the sequence of the detected human activities. Finally, by correlating the detected departure/arrival times of the vehicle with timetables/route maps collected from all the railway companies in the rider's country, it identifies potential routes that can be used for a trip. We demonstrate that the strategy is feasible through field experiments and extensive simulation experiments using timetables and route maps for 9,090 railway stations

of 172 railway companies.

Web side-channel attack to identify social account of a visitor is presented in Chapter 3. In this chapter, we demonstrate if an attacker observes data that the application leaks into the physical world, the Internet user's real-world identity will be leaked. Our attack leverages the widely adopted user-blocking mechanism, abusing its inherent property that certain pages return different web content depending on whether a user is blocked from another user. Our key insight is that an account prepared by an attacker can hold an attacker-controllable binary state of blocking/non-blocking with respect to an arbitrary user on the same service; provided that the user is logged in to the service, this state can be retrieved as one-bit data through the conventional timing attack when a user visits the attacker's website. We generalize and refer to such a property as visibility control, which we consider as the fundamental assumption of our attack. Building on this primitive, we show that an attacker with a set of controlled accounts can gain a complete and flexible control over the data leaked through the side channel. Using this mechanism, we show that it is possible to design and implement a robust, large-scale user identification attack on a wide variety of social web services. To verify the feasibility of our attack, we perform an extensive empirical study using 16 popular social web services and demonstrate that at least 12 of these are vulnerable to our attack. We have successfully addressed this attack by collaborative working with service providers and browser vendors.

Analyzing the inconsistency between behaviors and descriptions of mobile apps is presented in Chapter 4. In this chapter, we analyze applications that access privacy-sensitive data against human expectations. We focused on text description, which is written by a developer who has an incentive to attract user attention. Specifically, this chapter aims to address the following research question: What are the primary reasons that text descriptions of mobile apps fail to refer to the use of privacy-sensitive resources? To answer the research question, we performed empirical large-scale study using a huge volume of apps with our *ACODE* framework, which combines static code analysis and text analysis. We developed lightweight techniques so that we can handle hundred of thousands of distinct text descrip-

tions. We note that our text analysis technique does not require manually labeled descriptions; hence, it enables us to conduct a large-scale measurement study without requiring expensive labeling tasks. Our analysis of 210,000 apps, including free and paid, and multilingual text descriptions collected from official and third-party Android marketplaces revealed four primary factors that are associated with the inconsistencies between text descriptions and the use of privacy-sensitive resources: (1) existence of app building services/frameworks that tend to add API permissions/code unnecessarily, (2) existence of prolific developers who publish many applications that unnecessarily install permissions and code, (3) existence of secondary functions that tend to be unmentioned, and (4) existence of third party libraries that access to the privacy-sensitive resources. We believe that these findings will be useful for improving users' awareness of privacy on mobile software distribution platforms.

Finally, Chapter 5 presents the conclusions of this thesis.

Our research gives the research community a general insight into what to do to protect consumer privacy. Specifically, we provide important guidance on how to design secure sensor devices, web services, and app markets. Another aspect of our contribution is to implement countermeasures to real-world services. In fact, the countermeasures against a new threat we discovered have been adopted by global web services and web browsers. The privacy of hundreds of millions of users who use them is better protected than before.

Chapter 2

Sensor-based user location tracking

2.1 Introduction

Modern smart devices, such as smartphones, smart watches, and smart glasses, have powerful embedded sensors such as accelerometers, magnetometers, gyroscopes, ambient light sensors, and heart rate monitors. While these sensors are used to provide new user experiences, they also bring the new line of side-channel attacks [15–22].

Let us consider a new side-channel attack called SPS (sensor-based positioning system), which also exploits sensors of smart devices. The ultimate goal of an SPS attack is to estimate the location of a user by reading sensors but without using conventional geolocation methodologies such as GPS, cell tower signals, or WiFi. Clearly, achieving the goal is difficult, primarily due to the high degree of freedom of user mobility.

The goal of this work is to make the SPS attack feasible. To this end, we exploit the *spatio-temporal regularity of human mobility patterns* [23]; e.g., a person may use a fixed route on a transportation system for her/his commuting. Also, vehicles of transportation systems are generally expected to exhibit a temporal regularity unless they encounter operation problems such as natural disasters or rail accidents. We

expect that exploiting the regularity enables us to reduce the degree of freedom of human mobility.

With this approach in mind, we develop a novel proof-of-concept attack framework called *RouteDetector*, which targets the location of passengers of transport service. It aims to identify the route of your train trip (i.e., the sequence of train stations) by simply reading three hardware sensors – accelerometer, magnetometer, and gyroscope – which are all accessible from any apps without requiring any permissions. A unique technical concept of RouteDetector is that it makes use of not only data collected from multiple sensors embedded in a smart device, but it also leverages external data that can extract privacy information by correlating with collected sensor data.

The key technical components of RouteDetector can be summarized as follows: First, by applying a machine-learning technique to the data collected from sensors, RouteDetector classifies the activity of a user, e.g., walking, riding on a moving vehicle, or other status such as still. Next, using the sequences of the detected activities, RouteDetector extracts departure/arrival times of vehicle(s). Finally, RouteDetector correlates the extracted departure/arrival times of vehicle(s) with timetables/route maps of all vehicles and searches the potential mobility paths.

The key findings of this work are summarized as follows:

- Our field experiments using smart devices demonstrate that the RouteDetector framework can detect departure/arrival times of vehicles with errors smaller than six seconds on average.
- Our extensive simulation experiments using timetables and route maps for 9,090 railway stations of 172 railway companies demonstrate that given a sequence of departure/arrival times, RouteDetector can identify routes used for a trip by train, and the average number of identified routes becomes close to one if the number of stations used on a trip is more than six.

These findings support that the attack is feasible.

The rest of this chapter is organized as follows. Section 3.3 describes the threat models we assume for *RouteDetector* In section 2.3, we present the details of the

RouteDetector framework. Section 2.4 shows the results of performance evaluation. Section 4.6 discusses the limitations of *RouteDetector* and future research directions. We also discuss the possible counter measures against *RouteDetector*. Section 4.7 summarizes the related work. We conclude this chapter in section 4.8.

2.2 Threat models

The phase of side-channel attacks on mobile devices usually consists of three steps [24]. (1) A malicious application is spread through popular app markets. After installation, (2) it observes the leaking side-channel information. Based on the gathered information, (3) it uses the previously established model or templates to infer secret information. Similarly, our threat model assumes that a malicious software, which requires only a permission of Internet connection, is installed on the victim's device. We note that the malicious software does not require any other permissions, e.g. $ACCESS\_*\_LOCATION$, $ACCESS\_WIFI\_STATE$, and $ACTIVITY\_RECOGNITION$.

Namely, the application does not access location data resources such as GPS, identity of cellular base stations, and SSID of WiFi networks. The software keeps collecting sensor values and estimating the activities of the owner of the device; i.e., walking (running), moving on a vehicle, or other. Sequences of detected activities are periodically sent to the adversary's computer. The adversary's computer estimates the route of transportation by analyzing the sequences. Note that it is also possible that the user device computes the estimation of routes and sends the estimated results to the adversary. It is easy for an adversary to know the hardware model of the smart device; for instance, in the Android platform, by accessing the fields of `Android.os.Build` class, he/she can obtain the hardware information, such as brand, manufacturer, and/or model. He/she can also know whether a smart device is being held in someone's hand or is inside a bag by reading the ambient light sensor or proximity sensor. Because the threat model targets passengers on public transportation systems, it is not useful where no public transportation system is available. We also assume that the adversary knows the list of public transportation

Table 2.1 Summary of sensors. All sensors do not require permission for access.

Sensor	Type	unit	Description
accelerometer	physical	m/s^2	Acceleration applied to a device including the gravity.
linear acceleration	virtual	m/s^2	Acceleration applied to a device excluding the gravity.
magnetometer	physical	μT	Strength of geomagnetic field.
gyroscope	physical	rad/s	A device's rate of rotation.

systems that would likely be used by the victim. For instance, if a victim lives in a particular country, the adversary assumes that the victim may use any of railways available in that country. We also need to assume that the transportation system operates punctually; otherwise, RouteDetector's estimation may be inaccurate. We will study the issue in Sec. 2.4. Other limitations will be discussed in Sec. 4.6.

2.3 RouteDetector Framework

In this section, we present an overview of the *RouteDetector* framework (Sec. 2.3.1). Then, we describe the sensors we used for our analysis (Sec. 2.3.2). We then describe the key technical components of the *RouteDetector* framework; the detection of user activities in Sec. 2.3.3, detection of departure/arrive time sequences of vehicles in Sec. 2.3.4, and the extraction of candidate routes in Sec. 2.3.5.

2.3.1 Goal and Overview

The goal of the *RouteDetector* framework is to identify the route of a vehicle used by an owner of a smart device by reading the device's sensors. If a vehicle is a passenger train, a route is defined as a set of stations along a path. Figure 2.1 depicts the high-level overview that achieves the goal, together with the number of corresponding subsections that describe the technical details.

First, it reads values from sensors. As sensors, we picked up accelerometer, linear acceleration, magnetometer, and rotation vector. Details of data collection are described in Sec. 2.3.2. Next, we extract user activities from the collected sensor

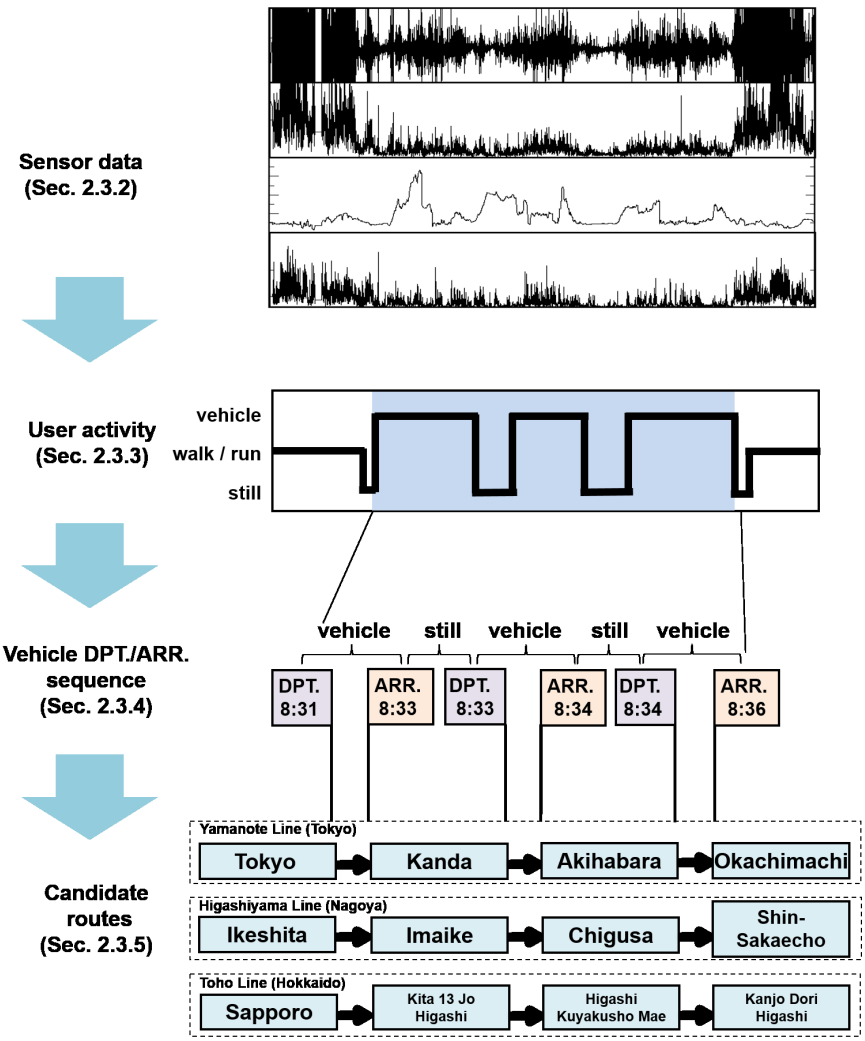


Fig. 2.1 High-level overview of the RouteDetector framework.

data. The user activities are defined as a set of three classes, *walking*, riding on a moving vehicle (*vehicle* in short), and *others*, which includes various activities such as standing, sitting, or sleeping. To this end, we pre-process raw sensor data so that we can apply a supervised machine-learning (ML) approach. As a supervised ML algorithm, we adopt random forest, which is known to achieve robust and good performance for multi-class classification tasks. Details of data pre-processing and ML application are described in Sec. 2.3.3. From the extracted user activities, we

can identify sequences of vehicle departure/arrival times. For instance, if we find a consecutive pairs of *vehicle* and *others*, it is likely that a user was on a vehicle. We can also consider cases in which a user made a transit. Details of detecting vehicle departure/arrival time sequence are described in Sec. 2.3.4. Finally, from an extracted vehicle departure/arrival time sequence, we search candidate routes, using timetables and railway route maps that cover the potential residential area of the victim, e.g., a country. We develop a fast algorithm that works in a breadth-first search manner. Details of extracting departure/arrival time sequence are described in Sec. 2.3.4.

2.3.2 Sensor Data

Of the available sensors embedded into a smart device, we adopt four sensors; accelerometer, linear acceleration, magnetometer, and rotation vector. Table 2.1 summarizes the sensors we used. Although we tested other sensors, such as an ambient light sensor, the data was not effective in detecting user activities. Note that the four sensors can be divided into two classes: physical sensors and virtual sensors. While the accelerometer, magnetometer, and gyroscope are physical sensors that read raw values, the remaining sensor, linear acceleration, is a virtual sensor whose values are computed based on physical sensors.

We developed an Android app that collects the sensor data. All the values are collected at a rate of 10 Hz, i.e., read 10 values per second. The app also has a function to generate labels that are used for supervised ML.

2.3.3 Detection of User Activities

Using the collected sensor data, we classify user activities into three distinct classes, *walk*, *vehicle*, and *others*. Note that *vehicle* refers to the status when a user is on a moving vehicle. If n user is standing on a vehicle, which is stopping at a station, his/her status is likely classified as *others*. We first pre-process raw sensor data in Sec. 2.3.3. Next, we apply a supervised machine-learning (ML) approach to the pre-processed data to detect user activities in Sec. 2.3.3.

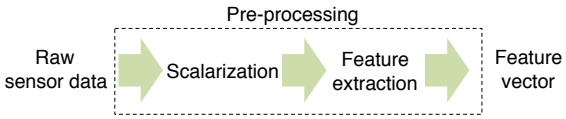


Fig. 2.2 Overview of data pre-processing.

Data Pre-processing

We apply several data pre-processing techniques to the raw sensor data. Figure 2.2 summarizes the data pre-processing scheme. First, to eliminate the effect of differences in the directions in 3D space, we compute a norm for each 3D vector; i.e., $a = \sqrt{a_x^2 + a_y^2 + a_z^2}$. Figures 2.3 (a) and (b) are examples of scalarized data. We then divide time series data into a set of blocks. A block consists of N samples for each sensor data; i.e., for each sensor data, a block b_i has data: $\mathbf{D}^{(i)}(a) = \{a_1^{(i)}, a_2^{(i)}, \ldots, a_N^{(i)}\}$. We experimentally set N as $N = 20$, which corresponds to 2 seconds length with the 10-Hz rate of sensor data sampling. For each block, we extract features that can be used to characterize the patterns of temporal variability for the three classes. To this end, we adopted simple metrics; i.e., mean, standard deviation, minimum, and maximum. Finally, we normalize the data by subtracting means and dividing by standard deviations. In summary, the time series data is divided into blocks, and each block consists of four features for four sensors, resulting in feature vectors with $4 \times 4 = 16$ dimensions.

Classifying User Activities

Using the pre-processed sensor data, we classify activities into three classes; *walk*, *vehicle*, and *others*. *Walk* represents the activity that of a person moving on foot; e.g., walking and running. *Vehicle* represents the activity that of a person riding in a moving vehicle; e.g., train and car. *Others* includes any other activities; e.g., standing, sitting, and sleeping. Although *others* can be further classified into subclasses, we did not need to do that because using these three classes are sufficient to achieve our attack.

As a classification scheme, we adopt the Random forest algorithm, which is an

ensemble learning algorithm used for classification or regression. In the training phase, the Random forest algorithm constructs multiple decision trees using randomly sampled data. In the classification phase, it predicts the most plausible class by taking the majority votes of the multiple decision trees. The good feature of Random forest is that it naturally achieves multi-class classification with a measure of score. We note that we also tested other supervised machine learning algorithms, such as SVM or logistic regression. It turned out that the differences in performance among the algorithms were not significant, but the Random forest algorithm worked best.

2.3.4 Detection of Departure/Arrival Time Sequences of Vehicles

Using the detected user activities, we extract sequences of vehicle departure/arrival times. Among the user activities, we are most interested in *vehicle* activity because the start/end of the activity corresponds with the departure/arrival, respectively. However, as shown in Fig. 2.3 (c), the predicted activities include some noise due to the inevitable classification errors. To reduce the effect of classification errors, we leverage the temporal correlation of the activities; i.e., once a user gets on a vehicle, it is likely that he/she stays on the vehicle for several minutes. Namely, we use the exponentially weighted moving average (EWMA) to account for temporal correlation of data.

Let A_n be the classified activity at block n, and $\mathcal{W}$, $\mathcal{V}$, and $\mathcal{O}$ be the set of blocks that are classified as *walk*, *vehicle*, and *others*, respectively. We define W_n, V_n, and O_n as

$$\begin{aligned} W_n &= \mathbf{1}_{\mathcal{W}}(A_n) \\ V_n &= \mathbf{1}_{\mathcal{V}}(A_n) \\ O_n &= \mathbf{1}_{\mathcal{O}}(A_n), \end{aligned}$$

where $\mathbf{1}_Y(x)$ is an indicator function that is defined as

$$\mathbf{1}_Y(x) = \begin{cases} 1 & \text{if } x \in Y \\ 0 & \text{if } x \notin Y. \end{cases}$$

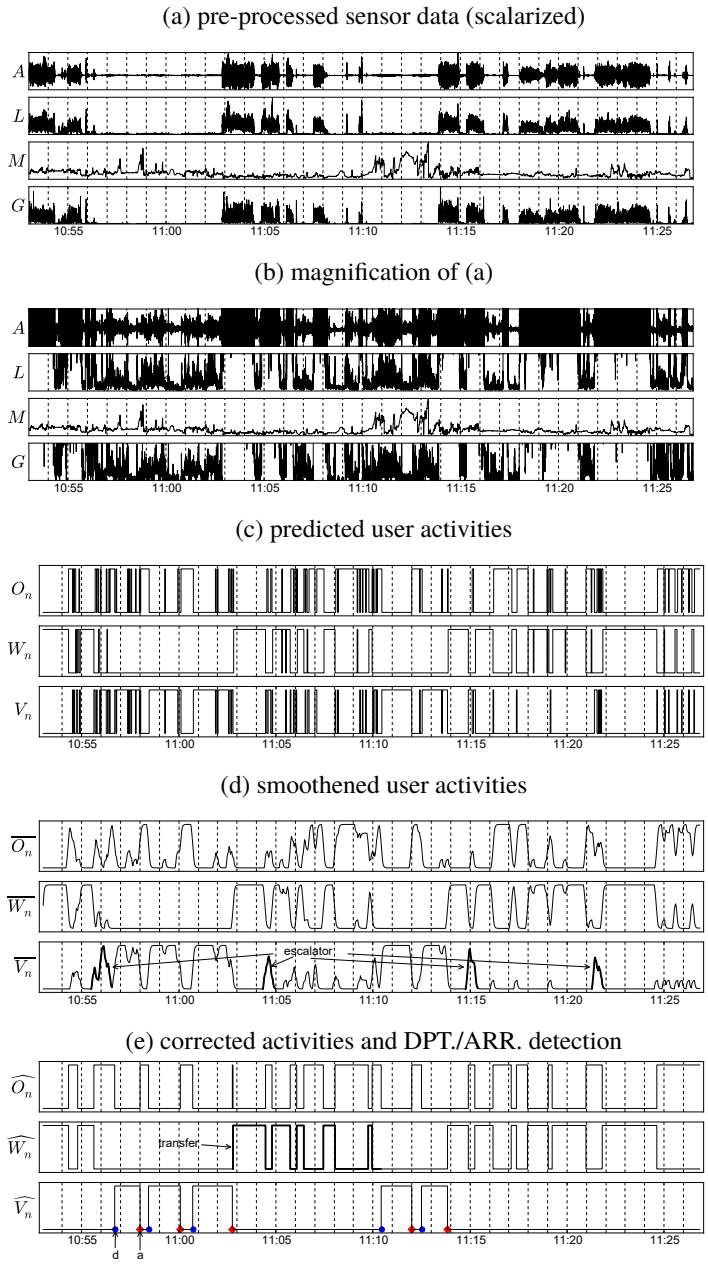


Fig. 2.3 (a): pre-processed sensor data, (b) magnification of (a) in Y-axis, (c) predicted user activities, (d) smoothened user activities, and (e) corrected user activities and departure/arrival times. In panels (a) and (b), A, L, M, and G represents accelerometer, linear acceleration, magnetometer, and gyroscope, respectively. In panel (e), circles/squares are detected departure/arrival times, respectively.

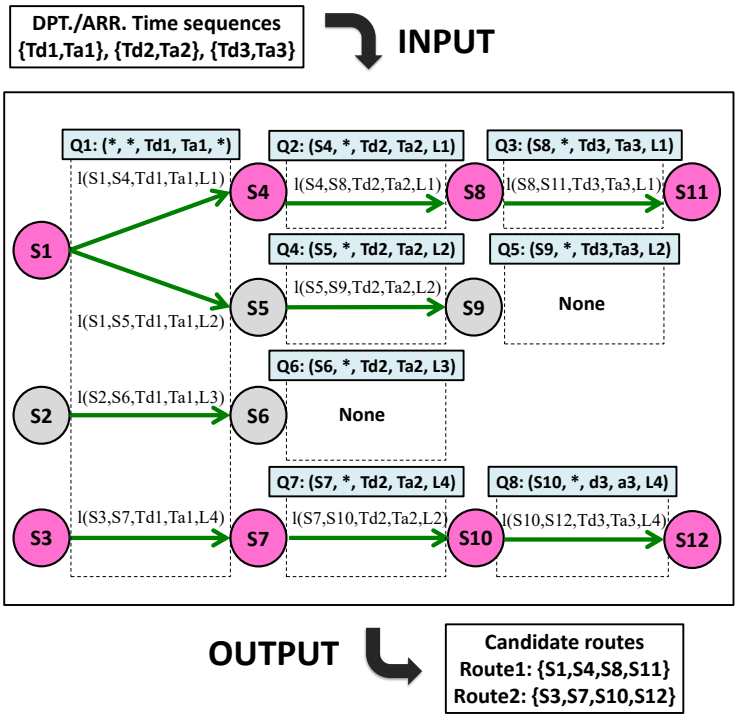


Fig. 2.4 Diagram of the route detection algorithm.

First, we compute the EWMA of V_n; i.e.,

$$\overline{V_n} = \lambda V_n + (1 - \lambda)\overline{V_{n-1}},$$

where $\overline{V_n}$ is EWMA and $0 \leq \lambda \leq 1$ is a constant parameter that determines the smoothing factor. If λ is close to one/zero, the EWMA has a larger weight on the last observation/past observations. The parameter λ is empirically configured, as we will show later. Although the EWMA introduces a certain time lag to the original data, the size of the lag was negligible, as we will show later. Using the EWMA, the classified activities are corrected, as

$$\widehat{V_n} = \begin{cases} 1 & \text{if } \overline{V_n} \geq 0.5 \\ 0 & \text{if } \overline{V_n} < 0.5. \end{cases}$$

Figure 2.3 (d) shows smoothened user activities with the EWMA.

Next, using the corrected activities $\widehat{V_n}$, we extract departure/arrival time sequences using the following algorithm, where τ is a threshold that determines the minimum length of time for a trip between two stations. In this calculation, we set $\tau = 60$ (seconds).

Algorithm 1 Vehicle DEP./ARR. time sequences detection algorithm.

D =**false** ▷ Initial state
for all $n = 1, 2, \ldots$ **do**
  if $\widehat{V_n} = 0$ AND $\widehat{V_{n+1}} = 1$ **then**
    $T_d = t_{n+1}$ ▷ t_n is time at block n.
    D =**true** ▷ A vehicle has been departured.
  if $\widehat{V_n} = 1$ AND $\widehat{V_{n+1}} = 0$ AND $D = 1$ **then**
    $T_a = t_{n+1}$
    D =**false**
    if $T_a - T_d > \tau$ **then**
      return T_a, T_d

We note that using blocks that were not classified as *vehicles*, i.e., $\{n; \widehat{V_n} = 0\}$, W_n and O_n can be corrected using the similar procedure. Tracking W_n and O_n is useful for detecting transferring lines; i.e., if we observe a sequence of classified activities such as *vehicle* (3 mins), *walk* (2 mins), *others* (4 mins), and *vehicle* (5 mins), it is likely that a person changed lines. Figure 2.3 (e) shows such an example. The victim first got on a train and got off the train after three stations. He/she then changed lines (see the area "transfer" shown in the graph of $\widehat{W_n}$), and got on the next train.

As we shall see later, the activity of riding an escalator could be misclassified as being on a *vehicle*, although a person may be using it for transferring lines. Such a misclassification can be safely removed with this heuristic. Figure 2.3 (d) and (e) show such an example where all the ground-truth escalator points, which were misclassified as "vehicle" by random forest, are successfully eliminated in the corrected user activities. The heuristics are also useful for eliminating other errors regarding activity detection.

2.3.5 Extracting Candidate Routes

Finally, using the extracted sequences of departure/arrival times, we estimate candidate routes. We formulate the estimation task as follows. Using railway route maps, we first create a single graph that consists of nodes (stations) connected by links (railroads). Next, using timetables corresponding to the railway route maps, we extend the graph so that it expresses temporal structure. Let us call the extended graph a "train graph." In a train graph, a link $l(A, B, T_d, T_a, L)$ expresses a vehicle that departures station A at time T_d and arrives at station B at time T_a; A and B are adjacent stations on line L. Note that we do not need to build/keep an entire train graph beforehand. Instead, we compile a set of all links and dynamically build subgraphs by applying our search algorithm to the set of links.

We use Fig. 2.4 to demonstrate how the algorithm of searching candidate routes works. In the example, we have the input departure/arrival time sequence of $\{T_d j, T_a j\}$ $(j = 1, 2, 3)$. Given the input, we first extract a set of links that satisfies $l(*, *, T_d 1, T_a 1, *)$ (Q1: query 1). In the example, we found four links; $(S1, S4, T_d 1, T_a 1, L1)$, $(S1, S5, T_d 1, T_a 1, L2)$, $(S2, S6, T_d 1, T_a 1, L3)$, and $(S3, S7, T_d 1, T_a 1, L4)$. For each link above, we recursively search the succeeding links. For instance, to find a link (vehicle) that departs station $S4$ at time $T_d 2$ and arrives at station X at time $T_a 2$ on line $L1$, we search a link that satisfies $l(S4, *, T_d 1, T_a 1, L1)$ (see Q2) and found $S8$ is the destination station. If we do not find any links that satisfy the given condition, we remove the paths from the search (see Q5, Q6). By continuing the above procedure, we can enumerate paths that satisfy the input departure/arrival time sequences; i.e., routes $\{S1, S4, S8, S11\}$ and $\{S3, S7, S10, S12\}$ in the example.

Finally, when we get multiple routes for a given time sequence, it is useful that we can sort them according to some metrics. To this end, we compute the popularity of routes, as follows: For each link consisting of a route, we compute the number of other links that share the same pair of origin/destination stations with that link. We then sum up the numbers along the links of a route and define the result as a score. If a route has a larger score, it means that a larger number of trains run on that route.

Table 2.2 Smart devices used for our analysis.

Device name (abbreviation)	Type	OS
HTC J Butterfly (HTC)	Smartphone	Android 4.1.1
Nexus 7 (Nexus)	Smart Tablet	Android 4.4.4

We adopt this score as a metric that expresses the popularity of a route.

2.4 Evaluation

In this section, we evaluate the performance of the RouteDetector framework. We first summarize the datasets we used for our analysis. Second, we evaluate the accuracy of the user activities detection scheme. We then evaluate the accuracy of departure/arrival time sequence detection. Finally, we evaluate the effectiveness of the candidate routes detection scheme.

2.4.1 Data

The data we collected for evaluation is broadly classified into two datasets. The first set consists of sensor data used for detecting departure/arrival time sequences. The second set consists of timetables and railway route maps that are used for building a train map, which is then used to search candidate routes for a given time sequence.

Sensor Data

Table 2.2 presents the two smart devices used for our analysis. As we shall see later, different hardware sensors generally exhibit different values when given the same input. Therefore, we need to train each classification model for each device. Details regarding to the differences in device hardware will be discussed in Section 4.6.

Table 2.3 summarizes the sensor data we collected. These data were measured across seven lines, operated by two railway companies. Four lines, Yamanote Line, Chuo Line, Keihin-Tohoku Line, and Saikyo Line, are operated by East Japan railway company. Three subway lines, Fukutoshin Line, Marunouchi Line, and Nanboku

Table 2.3 Sensor data collected for our analysis.

Data name	Device	Type	# stations	# lines	# blocks
HTC_H	HTC	H	57	5	12,007
HTC_B	HTC	B	29	1	2,561
Nexus_H	Nexus	H	29	1	2,543
Nexus_B	Nexus	B	54	5	8,576

Table 2.4 Statistics of the train map built from railway route maps and timetables. Number of links is taken from timetables for weekdays.

# railway companies	# lines	# stations	# links
172	597	9,090	2,277,397

Table 2.5 Numbers of labeled blocks used for evaluating performance of activity detection. All the labeled blocks are collected at the stations of Yamanote Line.

Data	*vehicle*	*walk*	*others*
HTC_H	609	1,327	510
HTC_B	691	1,360	510
Nexus_H	686	1,352	505
Nexus_B	602	1,304	505

Line are operated by Tokyo Metro. Of these lines, Yamanote Line is one of the busiest and most important lines that connect major stations in Tokyo. As shown in the table, we distinguish between two measurement types: a device held by hand (H) or located inside a still bag (B), which could be placed on the knee or on a rack. As we mentioned in Section 3.3, an adversary can distinguish the hardware of devices. He/she can also know whether a smart device is being held in someone's hand or is inside a bag by reading the ambient light sensor or proximity sensor.

Railway Route Maps and Timetables

While the coverage of collected sensor data is limited to a certain location, we use entire train services operated in Japan for building a train map. Table 2.4 summarizes

Table 2.6 Performance of detecting *vehicle* activity. ACC, FNR, and FPR are accuracy, false negative rate, and false positive rate, respectively.

Data	ACC (mean/std)	FNR (mean/std)	FPR (mean/std)
HTC_H	0.941/0.011	0.042/0.022	0.078/0.013
HTC_B	0.965/0.009	0.024/0.012	0.047/0.014
Nexus_H	0.943/0.013	0.041/0.014	0.074/0.021
Nexus_B	0.969/0.009	0.023/0.012	0.041/0.016

the data we collected. Note that a link $l(A, B, T_d, T_a, L)$ is defined in Section 2.3.5. We also note that if we can further specify the residential location of a victime, e.g., Kyoto area, the amount of data and candidate routes can be further reduced.

2.4.2 User activities detection

We applied our user activities detection scheme to the data shown in Table 2.5. The parameters of random forest were empirically optimized as $n = 50$ and $m = 4$, where n is the number of trees and m is the number of features used for each tree. To assess the generalization of the result, we employed 10-times, 10-fold cross-validation tests. We focused on the accuracy of detecting vehicles because it plays a crucial role in determining the departure/arrival time sequence. If a block of *vehicle* was incorrectly classified as *walk* or *others*, we defined it as a false negative. If a block of *walk* or *others* was classified as *vehicle*, we define it was false positive.

Table 2.6 summarizes the results. We noticed that classification accuracies are generally good in all the cases. We also noticed that measurement types of H, i.e., a device was inside a still bag, gave better accuracies. The result is intuitively natural because holding a smart device by hand may introduce motion noise.

2.4.3 Departure/Arrival Time Sequences Detection

Next, we applied our departure/arrival time sequence detection algorithm to the extracted user activities. For each dataset, we picked up departure/arrival time sequences of 30 stations. The 30 samples are divided into a training set and

Table 2.7 Absolute errors between detected times and observed (ground truth) times; departure (top) and arrival (bottom). m and σ are mean and standard deviation, respectively.

absolute errors of detected departure times.				
Data	min (sec)	max (sec)	m (sec)	σ
HTC_H	1.97	3.54	2.79	0.46
HTC_B	2.04	3.06	2.53	0.23
Nexus_H	2.33	7.94	4.60	1.84
Nexus_B	1.55	2.76	2.17	0.24
absolute errors of detected arrival times.				
Data	min (sec)	max (sec)	m (sec)	σ
HTC_H	2.52	6.75	4.13	1.18
HTC_B	1.71	4.63	3.21	0.77
Nexus_H	3.07	10.78	6.03	2.22
Nexus_B	2.22	5.16	3.43	0.80

a test set. Using the training set, the parameter of EWMA, λ, was optimized so that the difference between the detected departure/arrival time and observed departure/arrival time is minimized. Note that "detected" times are derived from sensors, "observed" times are manually labeled ones, and "scheduled" times are derived from a timetable corresponding to a train. To evaluate the performance, we employed 10-times, 3-fold cross-validation tests; i.e., 30 samples are randomly divided into 20 samples for a training set and 10 samples for a testing set, using different random seeds. Table 2.7 summarizes the absolute errors between detected and observed departure/arrival times. Note that observed departure/arrival times are not necessarily the scheduled times listed in timetables. The difference between the observed and scheduled times is shown in Fig. 2.5.

As we see, the detected departure/arrival times are close to the observed departure/arrival times. Maximal time differences are less than 3-11 seconds. On average, time differences are roughly smaller than 6 seconds. This amount of error has little impact on the overall estimation accuracy because our route detection, which makes use of train timetable, can allow up to 30 seconds of difference between the detected

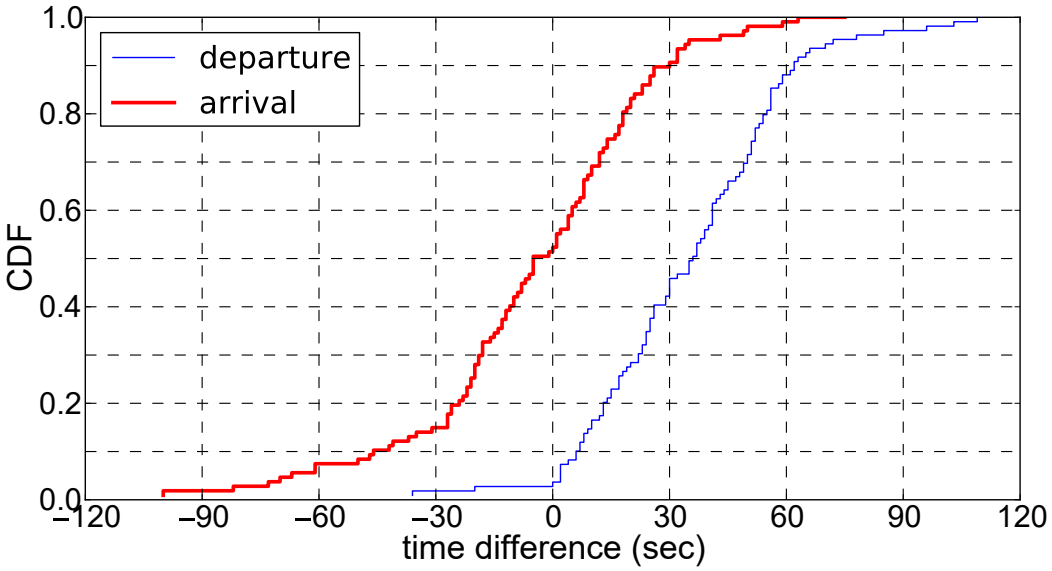


Fig. 2.5 Distributions of difference between observed and scheduled times. Departure times (top) and arrival times (bottom).

time and scheduled time.

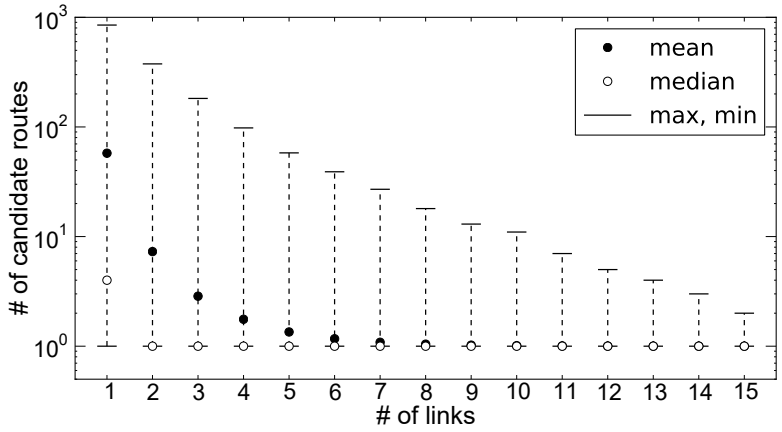
In addition, the observed departure/arrival times are also close to the scheduled times. Roughly 85% of trains depart within 60 seconds after the scheduled time has passed. Roughly 75% of trains arrived within 30 seconds around the scheduled time.

In summary, the detected departure/arrival times by the RouteDetector framework are close to the observed departure/arrival times, which are close to the scheduled times. In the next subsection, we show how we search routes given the detected departure/arrival time sequences. We also present several case studies in Sec. 2.5.

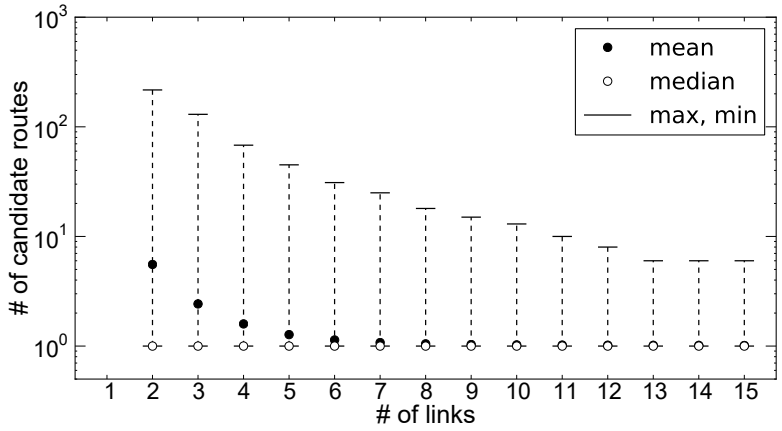
2.4.4 Candidate Routes Detection

While the evaluation of departure/arrival time detection scheme required empirical data, the evaluation of the candidate routes detection algorithm can be generalized

(a) no line changes



(b) with one line change



(c) with two line changes

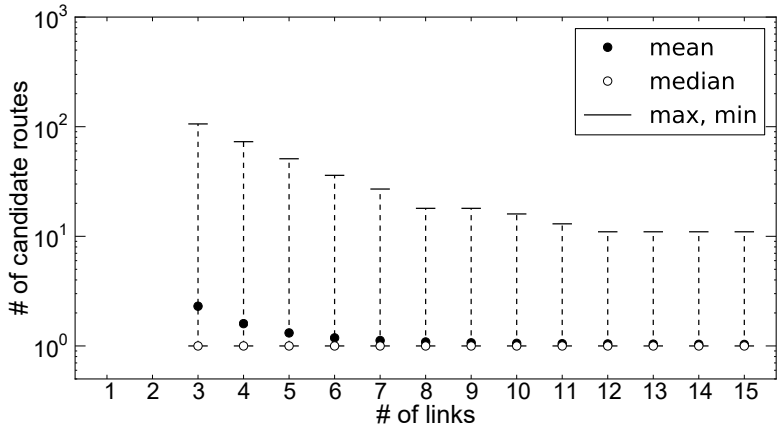


Fig. 2.6 Number of links vs. number of candidate routes.

by exploring paths on a train graph[*1]. Using the train graph constructed from the data shown in Table 2.4, we study the relationship between the number of links and the number of corresponding candidate routes. Figure 2.6 shows the results. (a) includes no line change, (b) includes one line change, and (c) includes two line changes, respectively. In (a), we can see that average number of identified routes becomes close to one if the number of stations used on a trip is more than six; i.e., if we observe more stations, the sequence of departure/arrival times become more unique. Even if the number of links is one, roughly 50% of time sequences T_d, T_a have less than four candidate routes. In addition, as shown in the panels (b) and (c), the RouteDetector can cope with a train trip with multiple line changes. We also see that if a train trip includes line change(s), the number of candidate routes becomes smaller, indicating line changes allows us to further narrow down the number of candidate routes.

Next, we study how quickly the search algorithm works. From the entire train graph, we first enumerate the routes whose lengths are less than 15 links, where we allowed, at most, two line changes. The number of enumerated routes was $6,404,455,757$. Using the C++ implementation of the algorithm that runs on a commodity PC, all these routes were searched within 74 mins. On average, a route was searched within 7.1 microseconds. Thus, the candidate routes detection worked quickly even though the scale of the train graph was huge.

2.5 Case study

In this section, we demonstrate the feasibility of the RouteDetector framework through the field experiments. Using sensor data collected from smartphone or tablet, we try to identify a route used for a trip. For brevity, we present three typical cases below. Figure 2.7 presents a map of lines used for the case study.

■Case 1 In this case, the train trip involved two lines, Yamanote line and Marunouchi line as shown in Fig. 2.7. Figure 2.3 presents the measured/derived

[*1]Because enumerating all the possible paths on a train graph could cause an explosion of states, we limit our search to the paths with lengths less than 15 stations.

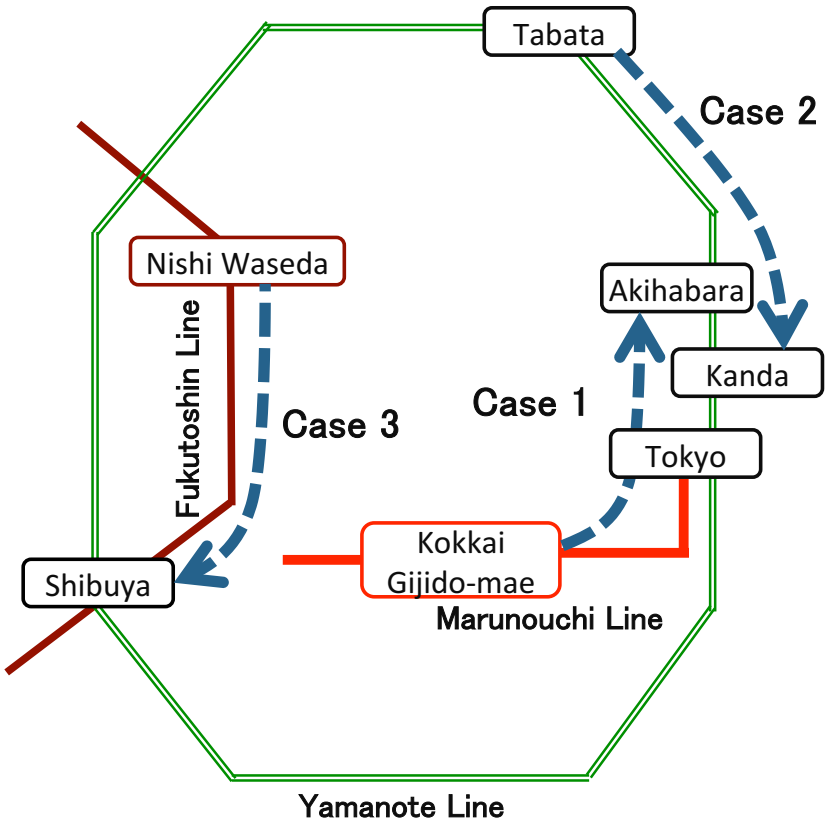


Fig. 2.7 Map of lines used for case study analysis.

data for the case 1. From Fig. 2.3 (e), we detected departure/arrival time sequence. The results are summarized in Table 2.8. As we see, all the detected departure/arrival times were correctly detected. Next, given this time sequence, we search the corresponding routes. The result is shown in Table 2.9, which shows two routes are identified. Of the identified two routes, the route #1 had higher score and was identical to the ground truth. Thus, the RouteDetector successfully detected a route used for a train trip from sensor data.

■Case 2 The case 2 was measured at Yamanote line. There was no transferring lines. The origin/destination stations were Tabata station and Kanda station, respectively. The trip involved 8 stations. Figure 2.8 presents the detected activities and departure/arrival time sequence. In this case, the detected departure/arrival times

Table 2.8 Detected/observed/scheduled times for case 1. Detected and observed times are rounded.

activities	detected	observed	scheduled
walking etc.		–	
departure	10:56	10:56	10:56
arrival	10:58	10:58	10:58
departure	10:58	10:58	10:58
arrival	11:00	11:00	11:00
departure	11:00	11:00	11:00
arrival	11:03	11:03	11:03
walking etc.		–	
departure	11:10	11:10	11:10
arrival	11:12	11:12	11:12
departure	11:12	11:12	11:12
arrival	11:14	11:14	11:14
walking etc.		–	

Table 2.9 Two identified routes for case 1.

No.	ground truth	route #1	route #2
1	Kokkai-gijido-mae	Kokkai-gijido	Edogawabashi
2	Kasumigaseki	Kasumigaseki	Gokokuji
3	Ginza	Ginza	Higashi Ikebukuro
4	Tokyo	Tokyo	Ikebukuro
		transfer	
4	Tokyo	Tokyo	Ikebukuro
5	Kanda	Kanda	Kanamecho
6	Akihabara	Akihabara	Sengawa
score	–	2,664	2,277

were correctly detected. Given the time sequence, a unique route was identified. The identified route was identical to the ground truth.

■Case 3 The case 3 was measured at Fukutoshin Line. Again, there was no transferring lines. The origin/destination stations were Nishi Waseda station and

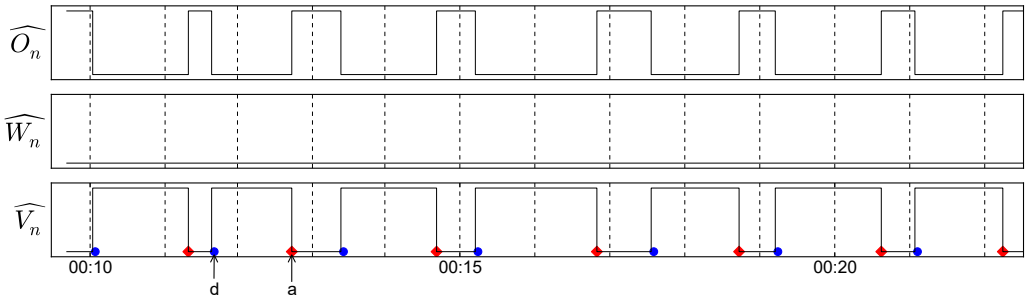


Fig. 2.8 Detected activities of the case 2.

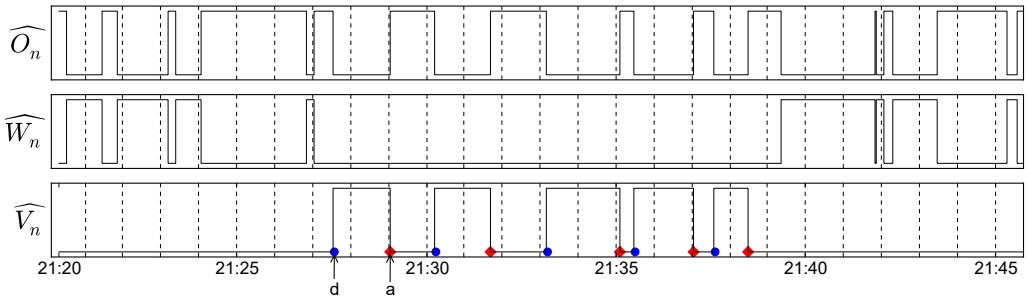


Fig. 2.9 Detected activities of the case 3.

Shibuya station, respectively. In this case, while the detected departure/arrival times were identical to the observed times, they were slightly different from the scheduled time; i.e., the train was delayed at the time of measurement. We will discuss the issue of train operation in the next section. Given the detected time sequence, no train route was identified from the train graph.

2.6 Discussion

In this section, we discuss several limitations of the RouteDetector framework. We also discuss countermeasures against the new threat brought by the RouteDetector framework.

Table 2.10 Detected/observed/scheduled times for case 3.

activities	detected	observed	scheduled
walking etc.		–	
departure	21:27	21:27	21:26
arrival	21:29	21:29	21:28
departure	21:30	21:30	21:28
arrival	21:32	21:32	21:32
departure	21:33	21:33	21:32
arrival	21:35	21:35	21:35
departure	21:35	21:35	21:35
arrival	21:37	21:37	21:37
departure	21:37	21:37	21:37
arrival	21:39	21:39	21:39
walking etc.		–	

2.6.1 Limitations

■Types of Vehicles While the target of this work was passenger trains, there are other types of transportation services, such as monorails or airplanes. If we can assume that vehicles are operated accurately according to timetable schedules, we may have a good chance to detect a route used for a trip.

We conjecture that the RouteDetector will not work well for automobile transport services such as public bus transportation because an automobile makes a stop irregularly on the street, e.g., a traffic light. Figure 2.10 shows the vehicle activity which we detected with the field experiment of the bus trip. We also marked the times waiting for traffic lights and stopping at bus stops. Hence we could not find a difference between them from their interval times, the RouteDetector failed to detect accurate times which the bus made a stop at bus stops. It may become distinguishable by using other hints such as the audio announcement for passengers. We leave the issue for future work.

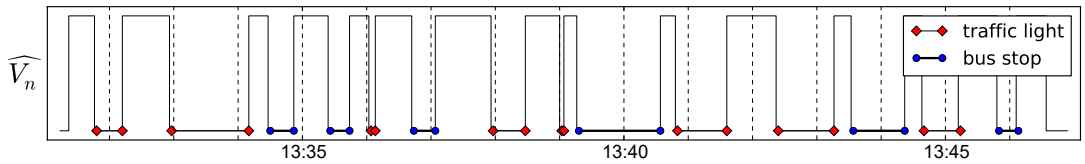


Fig. 2.10 Detected vehicle activities for a bus trip. The intervals marked with diamonds represent the waiting time for a red traffic light; the intervals marked with circles represent the waiting time at a bus stop.

■Train Operation Clearly, the success of the RouteDetector framework relies on the accuracy of the train operation. The detection accuracy may be limited in an environment where many trains tend to be delayed. For such a case, we need to study up to what amount of delay the attack works. To this end, we could artificially add a random delay and see how the framework reacts. We leave the analysis for our future work. We note that even in case of delay, some transportation systems provide information in real-time. Such information could be used to make the system more tolerant to delay.

We also note that by continuously targeting a victim, an adversary can obtain multiple observations, which likely include the correct estimations; e.g., commuting routes. Thus, by collecting many candidate routes used by a target, an adversary can figure out locations frequently visited by the target in a statistical way.

■Cross-Device Differences Our thread model assumes that an adversary knows the type of hardware to be attacked; i.e., he/she needs to have training data for detecting user activities for each device. In fact, we found that a random forest classifier trained to work with smartphone data did not work well for detecting the activities of tablet users. This observation suggests that a difference in hardware sensors is sensitive to the user activity detection scheme. One approach to this problem is to prepare training models for various devices. Another possible approach is to apply some data-processing techniques that can absorb the differences in the measurements of sensor values. We leave the issue for our future work.

2.6.2 Countermeasures

Let us discuss some ways to mitigate or eliminate the risk caused by the attacks using the RouteDetector framework. Michalevsky et al., presented *Gyrophone* [20], which is an attack that recognizes speech by reading gyroscope. They mentioned countermeasures in their paper that apply low-pass filtering to the raw samples provided by sensors. If certain pass frequencies are enough for most of the applications, the filtering can be done without negative effects. In addition, they mentioned that it should be controlled by permission mechanisms or certain explicit authorization by the user when certain applications require an unusually high sampling rate. In the same way, restricting access to raw sensor data and building some filtration mechanism that can remove sensitive information without sacrificing other functions would be promising approaches as countermeasures against the attack with RouteDetector. For instance, to build a pedometer app, a developer can use a specific API that can retrieve step counts, instead of reading row sensor values of accelerometer. Thus, building wrapper APIs that provide many useful functions, while hiding raw data, is a promising approach to thwart sensor-based side-channel attacks.

2.7 Related work

Techniques of sensor data analysis on mobile devices are mainly used for extending the range of application of mobile services, e.g., activity recognition and location-based services. On the contrary, attackers can expose user's privacy by using above similar techniques analyzing sensor data. We introduce techniques for both benign and malicious uses.

■Location inference and route tracing techniques There exist some studies which aim to identify user's position without GPS. An indoor positioning system (IPS) is presented as a solution to detect/navigate objects or people inside a building [25]. Instead of using GPS, IPS techniques make use of other information sources such as radio wave, acoustic signals, and optical signals. As an example of malicious use

of the positioning technique, Michalevsky et al. demonstrated that their developed *PowerSpy* application enables the attacker to infer the target device's location over those routes or areas by simply analyzing the target device's power consumption [22].

Also, a few works have focused on location identification by using the hardware-based sensors on mobile devices. Hua et al. demonstrate that user's location in subway systems can be tracked by using an accelerometer and information of interval of stations [26]. They use an ensemble interval classier built from supervised learning to infer the riding intervals of the user. On the other hand, the RouteDetector does not require training data for each station interval. Note that building the training data requires time-consuming manual effort because it requires collecting data along all possible paths. Our approach requires only railway route maps and timetables, which are both accessible open data. Thus, our approach can cover nation-wide locations without requiring large effort in building training data. Narain et al. showed that Android app can infer routes of automobile travel without any permissions [27]. They modeled this problem as a maximum likelihood route identification on a graph which is generated from a street map. What is common in these studies is that the accuracy of location identification may decrease when the targeted area is expanded. To demonstrate the feasibility in large scale, we evaluated a candidate route detection algorithm with a large railway map. Using timetables and route maps for 9,090 stations, we showed that the number of candidate routes greatly decreases.

In addition, there is a work that aims to extract more privacy sensitive information from GPS data. Tsoukaneri et al. developed *Comber* [28], which estimate user paths from anonymized mobility data. *Comber* is a system that identifies users and their corresponding paths given the completely anonymized GPS data as input.

■Device fingerprinting A device fingerprinting is other positive usage of sensors to identify and authenticate physical devices. Many studies reported that various IDs on a smartphone, e.g., IMEI (device ID), are easily stolen by malicious apps. To thwart ID-theft, Dey presented *AccelPrint*, which is a system that fingerprints based on the accelerometer, in order to identify devices without any specific ID or cookie [19]. Das et al. also discussed the feasibility of using sensors embedded in smartphones, i.e., microphones and speakers, to uniquely identify individual devices [21].

■Activity Recognition The *CenceMe* system developed by Miluzzo et al. [29] combines the inference of individuals' activity using sensors' information with sharing of it through social networking services. To classify activities (sitting, standing, walking, running) of individuals, the preprocessor of CenceMe calculates the mean, standard deviation, and number of peaks of the accelerometer readings along the three axes of the accelerometer. RouteDetector's activity detection scheme is similar to this one, but it is extended to capture the motion of vehicles. RouteDetector also uses other hardware sensors, such as a magnetometer and gyroscope, which also play a key role in improving detection accuracy.

The accelerometer sensor provides an attacker with other opportunities to build new attacks. Many attacks targeting motion sensors, i.e., accelerometers and gyroscopes, that are embedded in smartphones are inferring user inputs, e.g., passwords on touch-screens by monitoring readings collected from motion sensors [15–18].

■Sensor Access Control Although various kinds of sensor information contribute to extend and improve mobile computing and services, privacy issues have already been exposed as mentioned above. One of the most practical defenses is access control to sensor data. Unnecessary access by apps to sensor data should be controlled by OS or middleware on a device. *FlaskDroid* [30] and *ipShield* [31] are implemented as middleware on Android OS and provide fine-grain access control mechanism to resources including sensor information.

2.8 Conclusion

A novel, proof-of-concept side-channel attack framework called *RouteDetector* was introduced. The key idea behind the framework is to leverage *spatio-temporal regularity* of human mobility; i.e., we targeted passengers of train systems. Our field experiments demonstrated that the RouteDetector framework detected departure/arrival times of vehicles with errors less than 6 seconds on average. Our extensive simulation experiments using timetables and route maps for 9,090 railway stations of 172 railway companies demonstrated that the *RouteDetector* successfully identified routes used for a trip by train, and the average number of identified routes

became close to one if the number of stations used on a trip was more than six. These results quantitatively support that the attack is feasible.

Chapter 3

Web side-channel attack to identify social account

3.1 Introduction

The *Social web* has become ubiquitous in our daily lives. It includes not only popular social networking services such as Facebook and Twitter but also other forms of web services with social features, e.g., online services for video games such as XBox Live and online auction/shopping sites such as eBay. Social web services facilitate interactions between people with similar interests. The widespread adoption of social webs has increased not only the number of users per service but also the number of services used by each user. Mander [32] reports that Internet users have an average of almost seven social accounts.

Like many other web services, social webs have security and privacy concerns. What distinguishes social webs from other web services is that they have an intrinsic privacy risk; users are encouraged to share large amounts of personal/sensitive information on these services, e.g., personal photos, health information, home addresses, employment status, and sexual preferences. An attacker can collate various data from social web services to infer individuals' personal information. For example, as Minkus et al. [33] revealed, an attacker can recover a target's purchase history if s/he knows the target's eBay account. The purchases may include potentially sensitive

items, e.g., gun-related items or medical tests. To protect privacy, an eBay user may use a pseudonym for his/her account name; even in such a case, however, an attacker who can link an eBay account with an account on Facebook, which encourages users to disclose their real name, can infer the identity of the actual person who purchased the sensitive items on eBay.

In this study, we introduce a side-channel attack that identifies the social account(s) of a website visitor. The key idea behind our approach is to leverage *user blocking*, which is an indispensable mechanism to thwart various types of harassment in social webs, e.g., trolling, unwanted sexual solicitation, or cyber bullying. Because user blocking is a generic function commonly adopted by a wide range of social web services, an attacker can target various social web services. In fact, our attack is applicable to at least the following various social web services: Ashley Madison, eBay, Facebook, Google+, Instagram, Medium, Pornhub, Roblox, Tumblr, Twitter, Xbox Live, and Xvideos. Because having an account with some of the services included on this list could involve privacy-sensitive information, any account identification can directly lead to privacy risks.

Our attack leverages the user-blocking mechanism as a means of generating the leaking signals used for the side-channel attack*[1]. More specifically, we leverage the mechanism's inherent property that certain pages return different web content depending on whether or not a user is blocked from another user. Our key insight is that an account prepared by an attacker can hold an *attacker-controllable* binary state of blocking/non-blocking, with respect to an arbitrary user on the service, and this state can be retrieved as one-bit data through cross-site request forgery and a timing side channel when a user visits the attacker's website. We specifically refer to the property that enables this key action as *visibility control*, as an attacker is forcing another user to change how they see certain things in the system. Building on this primitive, we show that an attacker can use a set of controlled accounts to construct a *controllable* side channel, i.e, leaked data is completely under the attacker's control. Using this mechanism, we show that it is possible to design

*[1]More precisely, our side-channel attack is classified as a *cross-site timing attack* that will be described in Section 3.3.1.

and implement a robust, large-scale user identification attack mechanism on a wide variety of social web services. We note that the number of accounts required has a theoretically logarithmic relation to the number of users to be targeted, e.g., 20 attacker-prepared accounts are needed to cover 1 million users. The novelty of our attack is discussed further in Section 3.3.3.

We note that disabling our side channel, i.e., user blocking, requires careful assessment as it is a crucial function that is widely used on social webs. In Section 3.2.2, we discuss our analysis of the data for measuring the blocking behavior of more than 200,000 Twitter users [34] and revealed that 3,770 users have blocked more than 1,000 accounts. Our online survey also revealed that 52.3%/41.4% of Twitter/Facebook users have responded they have used the blocking mechanism before, and 92.4%/93.9% responded there should not be a limit on the number of blocks. These results suggest that neither disabling blocking nor posing a limit on it, is desirable from the viewpoints of the actual usage of the service and users' expectations. Furthermore, as we show in Section 3.5.3, limiting the number of user blocks per account would not be an effective countermeasure owing to our additional technique, *user-space partitioning*.

To verify the feasibility of our attack, we performed extensive empirical studies using 16 existing social web services. We found that 12 of these services are vulnerable to the attack. Using 20 actual accounts, we found that the attack succeeds with nearly 100% accuracy under a practical setting.

Our contributions can be summarized as follows:

- We demonstrate that the *user-blocking* mechanism, which is an indispensable function widely adopted in various social web services, can be exploited as the leaking signals for a side-channel attack that identifies user accounts.
- In addition to the side-channel attack, we develop several techniques to accurately identify users' accounts. We also reveal that this attack is applicable to many currently existing services. The attack has a high success rate of nearly 100%, and is high-speed, taking as short as 4–8 seconds in a preferable setting, or 20–98 seconds even in a crude environment with a large amount of delay.

- We discuss the principles, the practical aspects, and the limitations of this study, as well as some defenses against the attack.
- We have successfully addressed this attack by collaborative working with service providers and browser vendors.

3.2 Background: User Blocking

In this section, we first provide a technical overview of *user blocking*, which serves as a side channel used for the user identification attack. Next, we demonstrate that simply disabling/limiting this side channel is not a desirable solution against the attack from the viewpoints of actual usage and user expectations.

3.2.1 Technical Overview

User blocking is a means of blocking communication between two users. Note that some "blocking" mechanisms adopted by social web services are not *user blocking* per se but *message blocking*, e.g., "muting" or "ignoring". While user blocking rejects a person access to your account, message blocking filters out all the messages (or notifications) originating from that person. Even if a person is blocked with message blocking, this does not necessarily mean that they do not have access to your online activities. In this chapter, we will not focus on message blocking unless otherwise noted.

Social web services with *user-blocking* mechanisms have intrinsic web pages that change content depending on the status of the visitor, i.e., whether or not a visitor is blocked from another person. A typical example is a user profile data page, which provides information on a person such as a photograph (icon), a self-introduction, affiliation, recent posts/updates, etc. Figure 3.1 shows screenshots of some Facebook profile pages. In the non-blocked state, the user profile information is fully available; in the blocked state, these pieces of information are hidden. In addition to a user profile page, some social web services provide pages that reflect similar differences. A summary of such techniques is presented in Section 3.4.

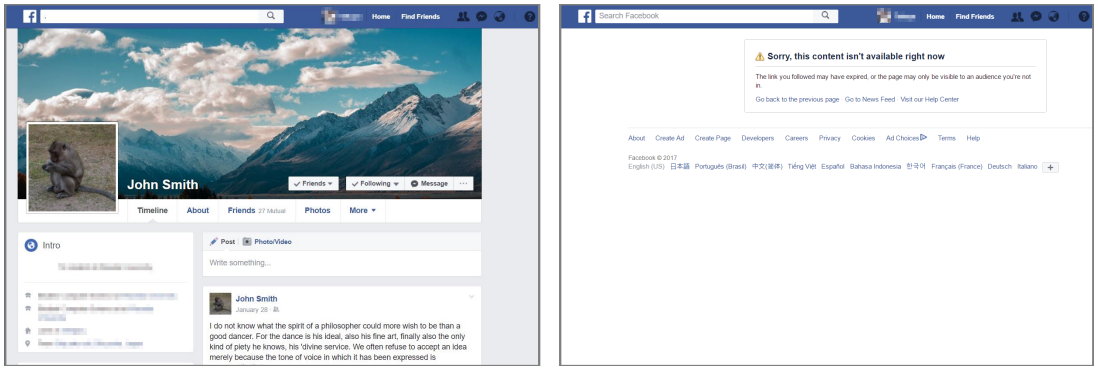


Fig. 3.1 Differences of appearance between non-blocking (left) and blocking (right) pages on Facebook.

To execute user blocking, a user typically clicks the "block" button set on the profile page of the person to be blocked or enters the account ID of the person in a text box shown on a dedicated page for user blocking. Even though official application programming interfaces (APIs) for performing user blocking are not necessarily provided on all social web services, to the best of our knowledge no services adopt a special mechanism, such as CAPTCHA, to prevent automated user-blocking requests. Therefore, it is currently easy to perform the large-scale user blocking necessary to implement our user identification attack by using a script that emulates authentic requests or a headless browser.

3.2.2 Usage and Expectations

In this subsection, we discuss how many accounts do people block on social web services and why they do so. To answer the "how many" question, we first present statistics derived from the data collected at "Blocked By Me" [34], a web service that displays a list of users a person has blocked on Twitter $*^1$. The data, comprising the number of blocked users for 223,487 unique accounts, were collected from

$*^1$The dataset was provided courtesy of Gerry Mulvenna on August 14, 2017. Thus, there are negligible differences from the figures listed on his webpage archived on August 20 [34]. Note that the entire set was anonymized to protect user privacy.

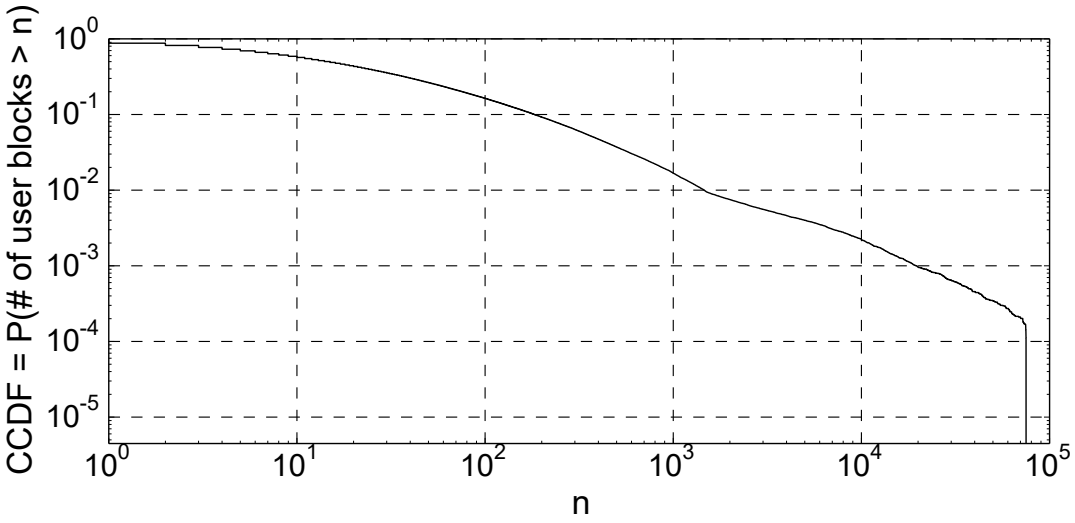


Fig. 3.2 Log-log CCDF of the number of user blocks per account on Twitter. Y-axis shows the fractions of accounts who are blocking n (X-axis) or more users. Mean value of the number of user blocks is 154.21.

March 2011 to August 2017. As an individual may have used the web service for several times during the measurement period, we adopt the maximum value of the numbers of blocked users measured for each person. Figure 3.2 shows the log-log complementary cumulative distribution function (CCDF) of the number of blocked users per account. It is seen that the distribution is heavy-tailed, indicating that, although the majority of users blocked a small number of other accounts (median = 15), a non-negligible number of users had to block a large number of other accounts. For instance, 3,770 users blocked more than 1,000 accounts. Note that the rate-limit of access to Twitter API truncates the number of blocked users at 75,000; thus, users indicated in the figure as having blocked 75,000 users are likely to have actually blocked more. Besides this upper bound, there were several groups of accounts having the same large number of blocked accounts. They may be using a shared block list to evade various harassments. As checking the content of such lists is not feasible, some users may have simply cumulatively added new accounts to their block lists. These insights account for the reason why several users have a large number of blocked users.

Table 3.1 Demography of the expectations survey.

	# respondents	Gender	Age (Years) 10–29 / 30–49 / 50–
Facebook	198	F:54 M:46 (%)	31 / 60 / 9 (%)
Twitter	170	F:56 M:44 (%)	41 / 51 / 8 (%)

Next, to answer the "why" question we recruited participants to take an online survey. As summarized in Table 3.1, the demography of the respondents shows that responses represent a diverse, cross-section of respondents. Key findings derived from the closed-ended questions are as follows: (1) 52.3%/41.4% of Twitter/Facebook users responded that they have used the blocking mechanism; (2) 92.4%/93.9% of Twitter/Facebook users responded that social web service should *not* limit the number of accounts a person can block on the service. This result indicates that users do not expect to have limitations on the number of blockable users. We also included the open-ended questions: "why do you block other users?" and "why do you think that there should be *no* limitation on the number of blocked users?" Typical answers to the first question include "do not want to read the unwanted messages/posts" and "not to be tracked by strangers/trolls/ex-friends/coworkers, etc." Typical answers to the second question include "there are a huge number of spam/bogus accounts" and "just adding unwanted users to the blocklist is easy to maintain."

The observations derived from the web service log analysis and the online survey imply that simply disabling our side channel, user-blocking, is *not* a desirable countermeasure against the threat from the viewpoints of actual usage of a service and users' expectations.

3.3 Attack Overview

In this section, we give a brief overview of the attack. We present the threat model and the attack flow with a concrete example. We also elaborate on the novelty of the attack and how it compares to some of the existing works in this area.

3.3.1 Threat Model

In this attack, the attacker's goal is to determine the social account of the visitors to her/his website. We present two possible attack scenarios under this goal. In the first, the attacker targets unspecified mass users in order to determine who visited the attacker's website, for the purpose of, e.g., marketing. In the second scenario the attacker targets a limited number of users with already known identities, such as their names or email addresses, and wants to determine their anonymous accounts which cannot be searched for using such identities. In both scenarios, the visitor's privacy is obviously breached, as the identity of the user or their private activity is revealed to the attacker without their consent. We further discuss the feasibility of building a target list in Section 3.7.

Our attack employs a *cross-site timing attack*, which is an attack that combines cross-site request forgery (CSRF) and a timing attack [35]. Cross-site timing attacks bypass the same-origin policy and enable an attacker to obtain information using the target's view of another site, i.e., in our context, the attacker can know whether or not the target user is blocked by the attacker-prepared *signaling accounts*. As we detail in Section 3.4, the status of blocked/non-blocked can be estimated from the time a web server of a social web takes to load a web content, or the round-trip time (RTT), of the profile page of a signaling account. As such, we make the following assumptions, which we will discuss in additional detail in Section 3.7.

Attack Trigger. We assume that the attacker can somehow induce their target to visit a malicious website. For example, the attacker uses malvertising techniques [36] or simply sends out email messages, in which case they can also link e-mail addresses to social accounts. Further details on this are discussed in Section 3.7.

Log-in Status. We assume that a target person has logged into the social web services, i.e., that cookies are enabled on the person's web browser. This assumption plays a vital role in the success of the attack because the logged-in status triggers the difference between views of profiles of blocking and non-blocking accounts. Because the majority of web services, e.g., Facebook, have an automatic sign-in

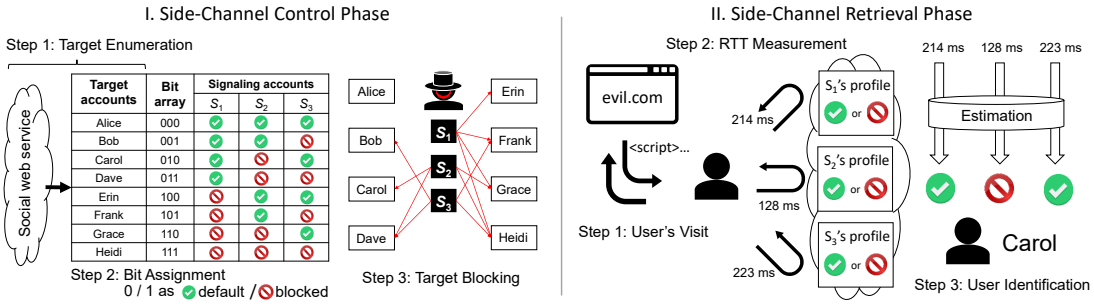


Fig. 3.3 Attack overview

option, we consider this assumption to be reasonable.

User Device. We assume that the target person uses a PC when accessing the malicious website. This premise covers more than 70% of social web service users [37]. Users of mobile platforms typically access social web services through dedicated mobile apps instead of the web interface provided for mobile browsers. Therefore, we cannot easily apply the attack to a mobile device.

3.3.2 Attack Flow and Example

As illustrated in Figure 3.3, our attack has two separate phases: the side-channel control phase and the side-channel retrieval phase. Below, we describe the steps in each phase with a concrete example. Note that some details are omitted for simplicity but will be described in later sections.

I. Side-Channel Control Phase

The purpose of the side-channel control phase is to construct user-identifiable side-channel data through user blocking. This phase is required just once before performing the attack.

Step 1. Target Enumeration: For a social web service of interest, the attacker first enumerates the users who will be the target of the attack. Let N be the number of targets. The attacker can target either mass (randomly sampled or even all) user accounts or a limited set of selected users (e.g., celebrities, high-level corporate officers) according to the attacker's purpose. Note that because this attack leverages

CSRF, whether the account is closed (e.g., private, protected) is not a concern as long as the account is blockable.

In the example, the attacker lists a small set of $N = 8$ users who will be the target of the attack. If the attack succeeds, the attacker will be able to identify the accounts of these eight users whenever they visit the attacker's website while logged onto the social web services.

Step 2. Bit Assignment: The attacker prepares m accounts on the social web service where m is a number satisfying $2^m \geq N$; these accounts are referred to as "signaling accounts" and denoted as S_i, $i = 1 \ldots m$. The attacker encodes a set of target users into bit arrays with length m, with the value of the i-th bit of each array corresponding to "block" (1) or "do not block" (0) by account S_i. The attacker can express a maximum of 2^m distinct target users, but at the cost of increase in m, the attacker can further add redundant bits to produce an error-correcting code.

In the example, the attacker prepares $m = 3$ signaling accounts, S_1, S_2, and S_3, with each target user is mapped into distinct bit arrays of length m, as shown in the table. All possible bit patterns are mapped to the users and there are no redundant bits.

Step 3. Target Blocking: The attacker controls the signaling accounts to block each target user according to the bit array. Note that the number of blocking that must be performed per signaling account is approximately half of the total number of targets, as shown in the figure. It is not difficult to see that this requirement can be controlled at the cost of adding more redundant signaling accounts, i.e., the block/non-block table in the figure will become more sparse.

In the example, S_1 is configured to block Erin, Frank, Grace, and Heidi, with the remaining four users left non-blocked (default). S_2 and S_3 are configured in a similar manner.

II. Side-Channel Retrieval Phase

The purpose of the side-channel retrieval phase is to identify the user utilizing the data retrieved through the timing side channel. This phase is executed every time a user accesses the attacker's website.

Step 1. User's Visit: When a user visits the web server under the control of the

attacker, JavaScript code is downloaded and is executed on the user's browser.

Step 2. RTT Measurement: The JavaScript code (as detailed in Bortz [35]) measures the time taken to load the profile of the signaling accounts by sending HTTP requests to each of these accounts. Note that, as this is a CSRF, the request is issued on behalf of the user's account. Special RTT measurements are also performed to determine the threshold value used in the next step, but we omit the details here.

In the example, the script issues HTTP requests to the profile page of each of the signaling accounts — S_1, S_2, and S_3 — and receives the measurements of 214, 128, and 223 ms, respectively.

Step 3. User Identification: The attacker then tries to identify the user from the measurements acquired in the preceding step. Because the time needed to load the profile of a blocking account exhibits a statistical difference from that needed to load the profile of a non-blocking account, the sequence of measured time samples can be used to build a bit array of "blocked" and "non-blocked" states. Once the bit array is recovered, the attacker does a lookup on the bit array map and identifies the user.

In the example, the measurements, 214, 128, 223 ms are compared against a threshold value of, say, 150 ms, and are determined to be non-blocked, blocked, and non-blocked, respectively. This result is represented as a bit array {010}, enabling the attacker to infer from the table that the user who visited the malicious site is Carol*[1].

3.3.3 Novelty of the Attack

While our attack is certainly novel overall, its conceptual novelty lies primarily in the side-channel control phase rather than in the side-channel retrieval phase, which can be implemented using many different existing approaches in addition to that adopted in our implementation [35]. The side-channel control phase is made particularly novel by its use of the underlying concept of *visibility control*, which

*[1] As we will detail in Section 3.5, when {000} is observed, it is still possible to distinguish Alice from non-target users by using two special accounts that do/don't block all the target users.

allows for the encoding and retrieving of arbitrary bits of data independent of what the side channel is. This flexibility inherently enables the attack to achieve account identification in a generic manner. By contrast, most similar methods that exploit browser side channels focus on stealing the content of a specific resource, limiting the acquirable data to that related to the targeted resource. Rather than studying such resource-specific side-channel acquisition methodologies, we questioned and exploited the design of general systems equipped with visibility-control features, e.g., user blocking. To the best of our knowledge, this concept has not been previously discussed in the literature despite its significant potential impact on nearly all major social web services currently operating.

We now compare our work to two of the major recent studies in this area. The goal of the first study was to retrieve various user data (e.g., age, contacts, search history) through several browser side-channel techniques [38]. The major difference between this work and ours is that it was somewhat focused on the development of individual techniques to acquire resource-specific side channels. Although this makes their methodology more powerful in the sense that it can even reveal a user's private information (e.g., search history), their methodology and goals were more service- and resource-specific. By contrast, the purpose of our work is to find user accounts and then link these with *all* available public information to which they are tied independent of the target resource used for sending side-channel data. Another similar study involved an attack based on browser history stealing [39], which, in the authors' words, shared a goal similar to ours of user identification or de-anonymization. This approach exploited the (now eliminated) mechanism allowing an attacker to infer a user's browser history to determine if the user belongs to certain groups based on the presence of access history to certain pages. Methodology-wise, the concept of repetitively identifying the groups to which a target user belongs, until to the point where the target can be uniquely identified, is conceptually similar to our approach. The main difference, however, is that our method allows for the construction of such groups in advance in an arbitrary manner. Thus, while our approach requires some initial setup effort, it has the advantage of being much more reliable in assuring identification (i.e., no ambiguity remains due to a lack of groups)

as long as the side channel can be correctly retrieved.

3.4 User-blocking Side Channel

This section aims to demonstrate that the differences between the time to load profile pages of blocked and non-blocked users can be used to perform a timing attack. In the following, we first look at the characteristics of the RTTs measured for blocked and non-blocked accounts. Next, we present several techniques that can increase the distinguishability of RTTs. Finally, after applying the RTT expansion techniques, we test whether the RTTs are statistically distinguishable using various social web services, which include popular social media such as Twitter and Facebook and other web services such as eBay and XBox Live.

3.4.1 Characteristics of RTTs

Here, we briefly describe the setup for our experiments. We executed a simple JavaScript code on a browser logged-in to a service with account A. The JavaScript issues GET requests to a page associated with an account which blocks A, and another page associated with an account that does not block A. Our objective is to see whether we can see the differences in the RTT measurements associated with these two types of accounts: blocking and non-blocking.

In the following, we characterize the measured RTTs using three social web services, Facebook, Twitter, and Tumblr as the representative examples. We study other services in the next subsection. Figure 3.4 shows the distributions of the measured RTTs[*1]. For Facebook, there is a clear gap between the RTT distributions for blocking and non-blocking accounts. For Tumblr, even though two distributions are closer, we see the difference between the distributions. We study whether or not this slight differences can be used as the timing side channel in Section 3.6. For Twitter, the distributions suggest that there is no sufficient difference to distinguish their

[*1] The results of this section were measured in early 2017. As we will explain later in Section 3.8.3, this attack is no longer work for the services that have adopted countermeasures through cooperation with us.

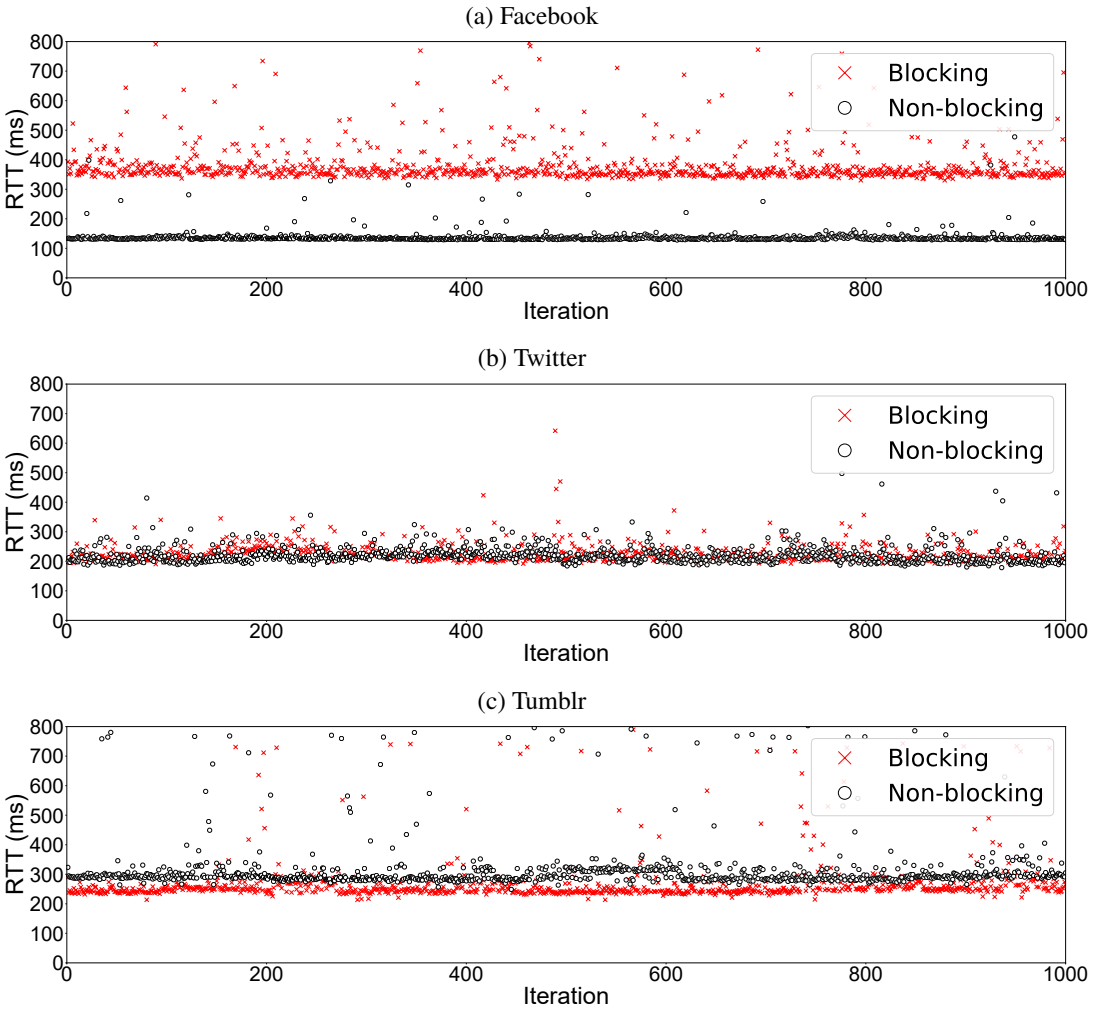


Fig. 3.4 Distributions of RTTs for blocking and non-blocking accounts.

RTT difference. Nevertheless, we have discovered that it is possible to intentionally amplify their RTT difference by posting more content to the profile page. More details on this will be described in the next subsection.

Note that, while we see longer RTTs for non-blocking accounts on Tumblr, we see longer RTTs for blocking accounts on Facebook. It is natural that the profile pages of blocking accounts are loaded quickly because the content of these pages may be lighter than those of the profile pages of non-blocking accounts. While not conclusive, we conjecture that this could be because Facebook does not utilize its

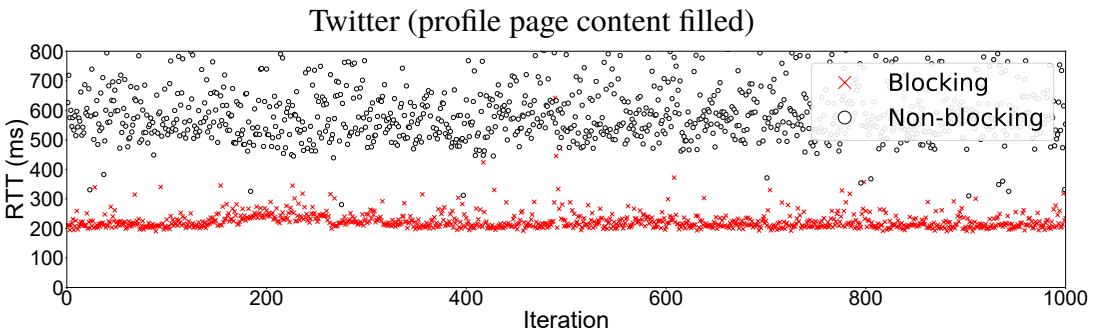


Fig. 3.5 Distributions of RTTs for blocking and non-blocking accounts, after filling the Twitter user profile page with content

server-side on-memory cache at all when generating content for the case of blocked. In either case, we can distinguish between the blocked and non-blocked states using the RTT measurements.

3.4.2 Improving RTT Distinguishability

We present three techniques that can make the differences in the RTTs more prominent, i.e., these are the ways to make the timing attack more successful.

Change of content size. The first technique is to place as much information as possible on the user profile pages of the signaling accounts. This technique can increase the time to load the profile page when the signaling account of the page is visible to the target, i.e., the signaling account does not block the target. We performed a simple experiment using Twitter. We prepared two Twitter accounts, one with the default setting and another with the maximum amount of content (texts and URL links) that appears on the profile page. Figure 3.5 shows the RTT distributions after filling the profile page with large amounts of content. Comparing this with Figure 3.4 (b) which shows the RTT distributions before adding the content, we now have a clear difference between blocked and non-blocked RTTs, suggesting that this technique can dramatically improve their distinguishability.

Use of different pages. Another technique is to make use of various pages other

than the user profile page. In many cases, the page subject to blocking is the user profile page, which displays the user's basic information or recent posts. However, depending on the service or their implementation, there are cases where observable differences do not appear on the profile page but do appear on other pages. For example, on eBay, a user cannot prohibit another user from accessing their profile page; however, a user can prohibit another user from bidding on the items they list. In other words, the content on the item page would yield a difference depending on whether the viewing user is blocked by the owner of the item. Leveraging this fact, by preparing an item beforehand and making the victim send requests to the item page instead of the profile page, the attacker would be able to observe the RTT difference required for the attack.

Similarly, Flickr does not prohibit a blocked user from viewing the blocker's profile page, but it does prohibit the blocked user from sending a message to the blocker. More specifically, there is a page for sending a messages to other users and, if the sender is not blocked from the receiver, a text area and a submit button are displayed on the page; however, if blocked, these objects are not shown and a warning message is displayed. Such a difference may also yield the RTT difference necessary for our attack.

In addition, some pages with AJAX-based implementation have a structure where after requesting and rendering the initial HTML content, they request additional content, e.g., a JSON content, from another URL using JavaScript's XMLHttpRequest. In some services, the blocked/non-blocked difference is only present in the JSON data that is acquired afterwards, instead of in the HTML content acquired first. The problem with this situation is that the RTT measurement script used for cross-site timing attacks does not actually render the acquired page content; therefore, the RTT of the content acquired afterward from JavaScript cannot be measured. In such cases, the attacker must directly send requests to the URL for the JSON data. In our investigation, we found that Tumblr and Xbox.com had this structure, but we were able to make the attack feasible by switching the request destination to the JSON URL instead of the HTML URL.

Table 3.2 Social web services with user-blocking mechanism. $\Delta_{0.05}$ shows the difference in 5-percent tile values for blocked/non-blocked RTT measurements. Dist. is the distinguishablity showing Y when the p-value less than 0.01. # of users are from various web resources as of May 2017

Service	Category	# users	$\Delta_{0.05}$	p-value	Dist.
Facebook	Social	1.96B	212 ms	<0.0001	Y
Instagram	Photo	700M	29 ms	<0.0001	Y
Tumblr	Microblog	550M	43 ms	<0.0001	Y
Google+	Social	540M	1,080 ms	<0.0001	Y
Twitter	Microblog	328M	312 ms	<0.0001	Y
eBay	Shopping	167M	589 ms	<0.0001	Y
PornHub	Porn	75M	9 ms	0.0034	Y
Medium	Forum	60M	332 ms	<0.0001	Y
Xbox Live	Game	52M	110 ms	<0.0001	Y
Ashley Madison	Dating	52M	8 ms	0.0097	Y
Roblox	Game	48M	98 ms	<0.0001	Y
Xvideos	Porn	47M	16 ms	<0.0001	Y
Quora	Forum	190M	5 ms	0.4561	N
Flickr	Photo	122M	1 ms	0.2678	N
DeviantArt	Art	65M	11 ms	0.0674	N
Meetup	Social	30M	9 ms	0.3878	N

3.4.3 Distinguishability of RTTs

We tested whether the RTTs for blocking and non-blocking accounts were statistically distinguishable. To this end, we leveraged the Mann-Whitney U test, which is a nonparametric statistical test used to compare differences between two independent samples; it tests whether a randomly selected value from one sample is less than or greater than a randomly selected value from another sample. For our experiments, we picked 16 popular social web services. For each service, we measured the RTTs between blocking/non-blocking accounts and the blocked account. We applied the Mann-Whitney U test and computed the p-values. The results are summarized in Table 3.2. The results show that all services have low p-values and imply that the distributions are distinguishable in 12 out of 16 services when the significance level is 0.01.

3.5 User Identification Attack

In this section, we first formulate the user identification attack, which works on the basis of the two building blocks, user-blocking and cross-site timing attack. The attack introduces two functions, encoding and decoding, which are the functions an attacker can arbitrarily set to map target users and leaking information (RTTs). Next, we describe the techniques we developed for the timing attack. Finally, we present two extensions of the attack. These extensions aim to make the attack more successful.

3.5.1 Formulation

Let m and N denote the numbers of the signaling and target accounts, respectively. We configure m as the minimum integer value that satisfies $2^m \geq N$. If an attacker wants to target one million of accounts, m is configured to $m = 20$.

In the setup phase, an attacker creates a table that maps target user accounts to bit arrays with a length of m. Let $U_i \quad (i = 1, \ldots, N)$ be the target user accounts. For each U_i, the table has a bit array entry, $B_i = \{b_1 b_2 \ldots b_m\}$, where $b_j \in \{0, 1\}$ corresponds to a bit. We refer to the rule that maps U_i into B_i as *encoding*, i.e.,

$$B_i = \text{encode}(U_i).$$

Next, we configure the signaling accounts, $S_j \quad (j = 1, \ldots, m)$ as follows. Let $\theta_{ij} \in \{0, 1\} \quad (i = 1, \ldots, N, j = 1, \ldots, m)$ be an indicator function that satisfies

$$\theta_{ij} = \begin{cases} 1 & \text{if } b_{ij} = 1\,, \\ 0 & \text{else,} \end{cases}$$

where b_{ij} is the j-th bit of the bit array B_i. Then, for each signaling account, S_j, the account is configured to block the user U_i if $\theta_{ij} = 1$. Because each bit takes the value $b_{ij} = 1$ with a probability of 0.5, each signaling account needs to block approximately $N/2$ target accounts. One may instantly come up with a defense that

poses a limit on the number of user-blocks an account can have. To thwart such a countermeasure, we propose a technique described in Section 3.5.3.

In the attack phase, the attacker sets up a malicious website and lets target users access it, following our threat model. As described in the previous section, using the timing attack, the website can secretly measure RTTs for the m of signaling accounts. Note that measurements can be parallelized to speed up the process. Let $\mathbf{R}_j = \{R_1, R_2, \ldots\}$ be the sequence of RTT measurements obtained for the signaling account S_j. Using the techniques that will be described in the next subsection, we estimate whether or not the target user is blocked by S_j. Let $\widehat{b_j} \in \{0, 1\}$ denote the estimate of the blocked/non-blocked (1/0) from the RTT measurements, i.e.,

$$\widehat{b_j} = \mathrm{est}(\mathbf{R}_j).$$

Using the entire estimates, we have the estimate of B as $\widehat{B} = \{\widehat{b_1} \ldots \widehat{b_m}\}$. Finally, we identify the target user using the table created in the setup phase; i.e.,

$$\widehat{U} = \mathrm{decode}(\widehat{B}).$$

In the next subsection, the estimation, $\widehat{b_j} = \mathrm{est}(\mathbf{R}_j)$, is described in detail.

3.5.2 Estimating Blocked/Non-blocked Status

Prior to the actual attack, we determine whether or not a visitor of the website has been included in the target list, i.e., we employ a membership test. To this end, we prepare the following two reference accounts: a *closed account*, which blocks all users included in the list of target users, and an *open account*, which does not block any users at all. We first measure the RTTs for each of the closed and open accounts. The measurements consist of k_0 trials for each account, where we use multiple trials because the decision based on a one-shot measurement may have errors due to jitter in the RTTs. The idea is to compare the measured RTTs for closed/open accounts to see if they are significantly different. If we observe a significant difference, we can conclude that the visitor has been listed and continue the attack; otherwise, the visitor has not been listed and the attack procedure is terminated.

To determine if the measured RTTs are for the closed or open accounts, we again leverage the Mann-Whitney U test. Because the computation of the U test is simple and lightweight, the membership test can be completed immediately after we collect the RTTs. In this study, we adopted a significance level of $\alpha = 0.01$. We also need to configure the parameter k_0. As shown in the next section, we empirically derived a conservative value of k_0 as $k_0 = 30$, which worked for various social web services.

After the attacker determines that the visitor is likely listed, the attacker moves to the next step. Let $C_{0.05}$ and $O_{0.05}$ be the 5th-percentiles of the RTT values measured for the closed and open accounts, respectively. We adopt the 5th-percentile as the threshold to eliminate outliers. Note that, even though we could use the minimum values for the RTTs as does the pathchar algorithm does [40], we observed that the RTTs could include small outliers, which could be caused by server-side mechanisms such as data caching or load balancing. These values are used as the thresholds to estimate the blocked / non-blocked state, i.e., for a measured RTT sequence for a signaling account S_j, we compute the 5th-percentile of $\mathbf{R}_j$ as $R_{0.05j}$. We do not necessarily make k, the number of trials S_j, equal to k_0. An attacker can adjust the k according to the his/her requirements for the trade-offs between accuracy and speed. If the obtained $R_{0.05j}$ is closer to $C_{0.05}$, the attacker estimates the visitor has been blocked by the signaling account S_j. Otherwise, s/he estimates the visitor has not been blocked by the signaling account; i.e.,

$$\widehat{b_j} = \begin{cases} 1 & \text{if} \quad |R_{0.05j} - C_{0.05}| < |R_{0.05j} - O_{0.05}|, \\ 0 & \text{else.} \end{cases}$$

By continuing this process for all $j \in \{1, \ldots, m\}$, the attacker can estimate the bit array of the visitor as $\widehat{B} = \{\widehat{b_1} \ldots \widehat{b_m}\}$. Finally, the bit array can be decoded into a user ID, $\widehat{U} = \text{decode}(\widehat{B})$, using the procedure we have shown in the previous subsection. Despite the simplicity of the procedure shown above, as we show later, it can estimate the closed/open states very accurately.

3.5.3 Extensions

Here, we introduce two extensions of the attack, *error-correction coding* and *user-space partitioning*, which aim to further improve the accuracy in noisy environments and to enhance the size of the target when the number of blocks per account is limited, respectively.

Error-correction Coding. Under a stable environment, accurately classifying a bit is not difficult since sufficient significant difference between blocked/non-blocked is present. This will also be shown later in Section 3.6. On the other hand, abnormal RTTs due to some irregular factors such as temporary server overload may lead to a bit-error. Needless to say, the infrastructures used in services such as those listed in Table 3.2 which host 30 million to 2 billion users tend to be quite resilient against such failures; nevertheless, we can still apply error-correction algorithm in order to eliminate even the slight possibility of identification failure due to noise.

In this chapter, we adopt a Reed-Solomon code, which has a high error-correction capability and is relatively easy to implement. In fact, as we will demonstrate later, the use of the Reed-Solomon algorithm actually contributes to improving the estimation accuracy in a noisy environment. Note that other error-correction algorithms could be used for this purpose. To select the most suitable error-correction algorithm, one must take into account several factors such as the error probability distribution, the error characteristics such as bursts, and the requirements of the available computing resources. In this chapter, we are focused on the proof of concept; therefore, we consider choosing the best error-correction algorithm to be out of the scope of this study.

The Reed-Solomon algorithm can correct up to $K/2$ symbol errors, where K is the number of redundant symbols and r (bits) is the size of the symbol. Because the number of bits initially allocated to each user is m, the number of signaling accounts that needs to be prepared by the attacker is $m + rK$, i.e., the attacker needs to prepare an additional rK extra signaling accounts. In the setup phase, the attacker first encodes the bit arrays allocated to each user using a Reed-Solomon encoder, and

then blocks the accounts from the signaling accounts according to the bit values of the newly encoded bit array. In the attack phase, by decoding the bit arrays obtained via the cross-site timing attack using the Reed-Solomon decoder, the attacker can obtain an error-corrected bit array.

User-space Partitioning.

As described in Section 3.2.2, simply enforcing a limit on the number of blocks would violate a user's right to block and may result in a serious degradation of the service quality. For services that still enforce a limit despite this negative impact, the technique shown below would be effective. Letting this limit to be L, the number of candidate target users covered for identification is also limited to L when using the procedures we have introduced up to this point. To lift this limitation, we can employ a technique we call *user-space partitioning*, which in this case splits candidate users into S user spaces each containing L users, allowing us to cover up to LS users in total.

In the setup phase, for each user space $j \in \{1, \ldots, S\}$, an attacker prepares a reference account that blocks all users belonging to the j-th user space and the $\lceil \log_2 L \rceil$ of signaling accounts that are used to map the targets in the space. We also prepare the two reference accounts, the closed and open accounts, which are used as the basis of the RTT-based blocking/non-blocking estimation. In total, the number of signaling/reference accounts required is $S\lceil \log_2 L \rceil$ and $S + 1$, respectively.

In the attack phase, the attacker (1) identifies which user space the target user belongs to and then (2) identifies the target in the user space. In step (1), as in the procedures described in the previous subsection, for each of reference account, k requests are launched to determine the user space to which the target belongs. Note that the RTT values obtained here can be reused as the training data in step (2). In step (2), for each of the L users in the user space found in step (1), the same identification process is performed as explained earlier. Note that, because we use a different set of signaling accounts for each user space, the request URL for the cross-site timing attack must be changed depending on the outcome of step (1); however, this can be handled with conditional branches in the JavaScript code.

3.6 Field Experiments

In this section, we perform the field experiments. We first evaluate the key success factor of the attack – RTT measurement, which plays a vital role in classifying blocked/non-blocked status using the cross-site timing attack (Section 4.4.3). Next, we evaluate the feasibility of our user identification attack; namely, we study the identification success rate (Section 3.6.2) and time to complete the attack (Section 3.6.3).

3.6.1 Accuracy of Bit Array Estimation

Due to space and time constraints, we evaluated the accuracy using RTT values experimentally measured for the following three services: Facebook, Twitter, and Tumblr. As shown in Table 3.2, these services have the top number of users and, at the same time, had no limitations such as the limit on the number of blockable users at the time of the experiment. In addition, as mentioned in Section 3.4, each of these three services had different characteristics in the blocked/non-blocked RTT difference: relatively large, medium, and small, respectively.

The experiment was conducted by executing the JavaScript on a consumer laptop PC and measuring the RTT.

We argue that the user's environment, i.e., network conditions and web browser engines, does not affect the success or failure of our attack. More precisely, our attack implementation absorbs the differences in the processing time caused by network jitters or the performance of the rendering engines. To demonstrate the assumption, we used the following three different network environments: wired LAN, Wi-Fi, and tethering. The wired LAN and Wi-Fi were connected to a commercial Internet service provider, and we assumed that this is the environment of PC users who are the main targets of our attack. To prove that our attack is feasible even in crude environmental conditions, we also used a tethering network hosted on an Android device connected to a 4G network provided by a mobile network carrier. In addition, we installed three of the most used browsers in the world, Google Chrome (v58),

Mozilla Firefox (v53), and Microsoft Internet Explorer (v11). Note that trying all combinations was difficult due to space limitations. Unless otherwise noted, we used Google Chrome with a wired LAN.

Membership Test. We first tested the accuracy of the membership test. We measured the RTT for each of the closed and open accounts. As mentioned earlier, the measured RTT values are used for (1) the membership test and (2) deriving the thresholds for the bit classification, which will be described later. Note that an attacker needs to calibrate the thresholds before launching the attack because the RTT values depend on the geographical location and network environment.

We repeated the following experiment 100 times. While logged on to a target and non-target account, we launched k_0 trials for each account and decided whether or not the account was included on the list by applying the Mann-Whitney U test. We refer to the true positive rate (TPR) as the ratio of correctly deciding that the target was included in the target, and the true negative rate (TNR) as the ratio of correctly deciding that the target was *not* included on the target list.

Figure 3.6 shows the relationship between k_0 and TPR/TNR. When k_0 is small, we have a small number of samples to estimate the states. Nevertheless, thanks to the strong distinguishability of the RTT distributions, TNR was 0.97 for all k_0, i.e., there were very few false negatives, which are events where the target account was estimated as *not* being listed. Second, for TPR, we saw degradation in the accuracy when k_0 was small, especially for Tumblr. As k_0 increases, however, the TPR approaches 1.0. When choosing the value of k_0, it is preferable that the accuracy is consistent and that we see a sufficient difference in the samples. If k_0 is large, the accuracy will increase but the number of trials will also increase and the time needed for an attack would become too long. In this experiment, we empirically chose $k_0 = 30$, which achieved perfect estimations for all the services. We will use the values of $C_{0.05}$ and $O_{0.05}$ calculated from this k_0 as the thresholds used in the next step.

The measured RTT values can be affected by various external factors such as network latencies or the type of browser. We studied how these factors affected the TPR/TNR. Table 3.3 shows the results. The number of trials was set to $k_0 = 30$.

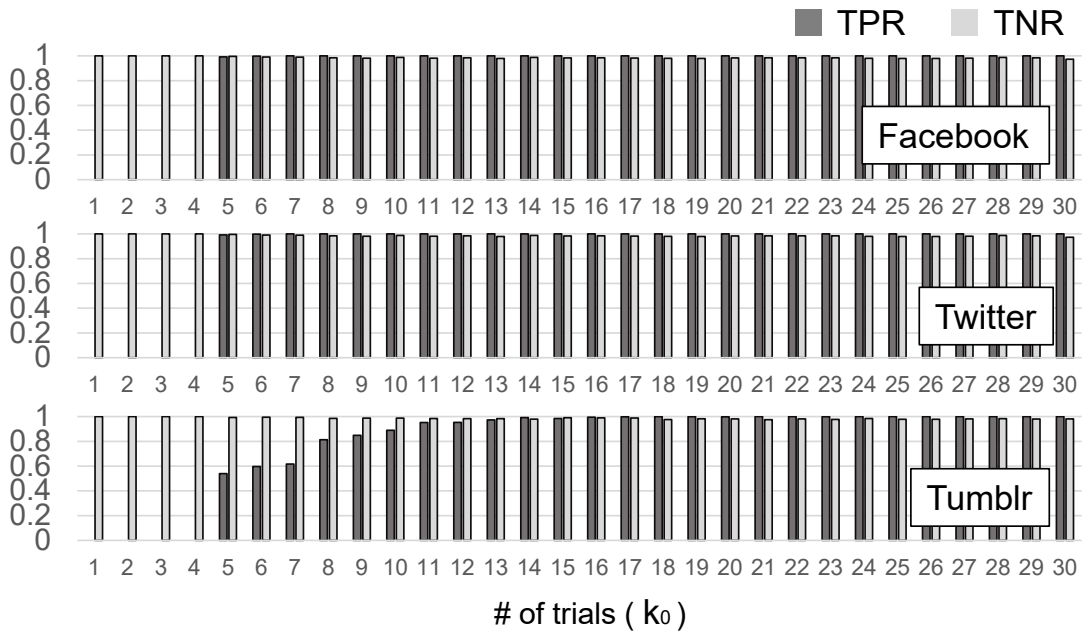


Fig. 3.6 Relationship between the number of trials (k_0), and TPR/TNR.

Table 3.3 TPR and TNR for under various conditions.

	Facebook		Twitter		Tumblr	
	TPR	TNR	TPR	TNR	TPR	TNR
Chrome/Wired	1.00	0.99	1.00	0.98	1.00	0.99
Wireless	1.00	0.98	1.00	0.98	1.00	0.99
Tethering	1.00	0.98	1.00	0.97	1.00	1.00
Firefox	1.00	0.98	1.00	1.00	1.00	1.00
IE	1.00	0.98	1.00	0.98	1.00	1.00

Single Bit Classification. Next, we evaluated the accuracy of classifying a single bit into blocking or non-blocking. Again, we used three social web services, Facebook, Twitter, and Tumblr. For each service, we performed k trials of RTT measurements for each of two signal accounts with blocked/non-blocked states. We continued this process for 100 times and took the mean values of the following metrics. We refer to the true blocking rate(TBR)/true non-blocking rate(TNBR) as the rate

Table 3.4 Accuracy of classifying a single bit for Wired(top), Wi-fi(middle), and Tethering(bottom)

	Facebook		Twitter		Tumblr	
k	TBR	TNBR	TBR	TNBR	TBR	TNBR
1	1.00	0.98	0.99	0.99	0.67	0.99
3	1.00	1.00	1.00	0.99	0.89	0.99
5	1.00	1.00	1.00	0.97	0.95	0.98
10	1.00	1.00	1.00	1.00	0.98	1.00
20	1.00	1.00	1.00	1.00	1.00	1.00
30	1.00	1.00	1.00	1.00	1.00	1.00
1	1.00	0.98	0.98	0.99	0.84	0.99
3	1.00	1.00	1.00	0.99	0.98	1.00
5	1.00	1.00	1.00	0.99	1.00	1.00
10	1.00	1.00	1.00	1.00	1.00	1.00
20	1.00	1.00	1.00	1.00	1.00	1.00
30	1.00	1.00	1.00	1.00	1.00	1.00
1	1.00	0.97	0.98	0.99	0.68	0.99
3	1.00	0.99	1.00	0.98	0.92	0.99
5	1.00	0.98	1.00	0.97	0.98	1.00
10	1.00	1.00	1.00	1.00	1.00	1.00
20	1.00	1.00	1.00	1.00	1.00	1.00
30	1.00	1.00	1.00	1.00	1.00	1.00

of correctly detecting the blocking/non-blocking user as a blocking/non-blocking user, respectively. Table 3.4 shows the results. When $k \geq 20$, the detection becomes perfect for all the three services. Moreover, in a stable environment such as Facebook/Wired, the classification succeeds perfectly even with $k = 3$.

3.6.2 Attack Success Rate in the Wild

We now show the result of our experiment conducted in an environment imitating an actual attack scenario in the wild. We set the length of a bit array to $m = 24$, which can cover over 16 million users. In addition, we applied a Reed-Solomon code with a block length of 4 bits with eight redundant bits, which enables it to correct one block of error. According to the above setting, we prepared 34 accounts in total, which included 32 signaling accounts, a closed account, and an open account, with the appropriate blocking done against the users on the target list.

Regarding the targets, we assigned a random bit array of length 24 to each of the 10 social accounts we actually own. We encoded these bit arrays using the Reed-Solomon code and calculated the bit arrays assigned as the redundant bits. We prepared 10 additional accounts which are not included in the list. For each of the 10 accounts on the target list and the 10 accounts on the non-target list, we logged in to and accessed the attacker's website and evaluated if the account was correctly identified. We repeated the visit two times per account, resulting in a total of 40 identification trials.

As the parameters for the number of trials, we selected $k = 30$, which we experimentally determined yielded good accuracy. The service and network environment pairs we chose were Facebook/Wired LAN, Twitter/Wireless LAN, and Tumblr/Tethering. We refer to the TPR as the rate of correctly identifying a target to be included on the list, and the TNR as the rate of correctly identifying a non-target to be not included on the list.

As mentioned above, we conducted the experiment twice with each account using 10 target accounts and 10 non-target accounts. Note that the denominators of TPR and TNR are 20. In addition, of the users who were identified as being included on the target list, we refer to the identified rate (IDR) as the rate of correctly identifying the user without the error-correction code, and refer to the identified rate with error correction (IDR/EC) as a similar figure but with error correction. In Table 3.5, we show the classification accuracy we obtained in this experiment.

The result shows that the experiment succeeded with extremely high accuracy.

Table 3.5 Accuracy of the User Identification Attack.

	Facebook/wired	Twitter/WiFi	Tumblr/tethering
TNR	1.00 (20/20)	1.00 (20/20)	0.95 (19/20)
TPR	1.00 (20/20)	1.00 (20/20)	1.00 (20/20)
IDR	0.95 (19/20)	1.00 (20/20)	1.00 (20/20)
IDR/EC	1.00 (20/20)	1.00 (20/20)	1.00 (20/20)

This was expected from the good results we obtained from the experiments in Section 3.6.1. For Facebook/Wired, there was one failure case which identified the target as a wrong user. Examining the network log for this case revealed that some requests to one of the signaling accounts had returned 502 response code due to temporary server error. Our script measures the RTT even if an error code is returned, but since no content is returned, the response time would not likely be the one desired. This occurred with 3 of the requests over only 1 second of duration, but the RTT value had dropped to about 1/5 of the true RTT which was enough to cause a bit error. Nevertheless, applying the error-correction algorithm, we were successfully able to correct this bit which resulted in the success of identification. Note that, because we adopt the 5th-percentile, our attack is resilient to outliers which are too late, but it is prone to those which are too early.

Another case of failure was for Tumblr/Tethering, where a non-target user was incorrectly identified as a target. This is a rare case where a significant difference of around $p < 0.01$ happened to occur when comparing the two sets of 30 non-blocked requests. This example also benefited from the error-correction algorithm; without error-correction this visitor would have been identified as another user, but with Reed-Solomon code, although the error was not correctable due to too many errors, the error was still detectable. In such a case, we can still prevent mis-identification by concluding that the membership test failed and restarting the test.

3.6.3 Time to Complete the Attack

The shorter the time required for the attack, the more feasible the threat is. While the total number of requests can be calculated beforehand, the time required to complete these trials is dependent on the actual RTT; therefore, it needs to be evaluated experimentally. Figure 3.7 shows the relationship between the number of trials and the required time for each service.

The "upper bound" value shown for each service assumes the request with whichever has the larger of the blocked/non-blocked RTT values, that is, it assumes the case with the longest time needed for identification; i.e., it is the worst case. Conversely, the "lower bound" value assumes the request with whichever has smaller value of the two, that is, it assumes the case with the shortest time needed for identification; i.e., it is the most optimistic case. The number of trials issued in parallel was set to 6, which is the default maximum number of concurrent connections allowed on major browsers such as Chrome, IE, and Firefox.

The total number of requests needed to make an m-bits decision, or in other words, to identify the target within 2^m users, is $mk + 2 \times 30$ when $k_0 = 30$. For example, for $m = 24$, or targeting 16 million users, the total number of requests needed is 780 when $k = 30$. This would require 20–50 seconds for Facebook, 32–98 seconds for Twitter, and 64–68 seconds for Tumblr. According to Table 3.4, in the case of Twitter, we have sufficient accuracy even with $k = 10$. The number of necessary trials is 300 with this setting, and the time required is 12–37 seconds. Moreover, we can observe that we can achieve sufficient accuracy even with $k = 3$ on Facebook. The total number of requests is 132 which only takes 4–8 seconds.

3.7 Discussion

In this section, we discuss the attack's principle, practical aspects, known limitations, and ethical considerations.

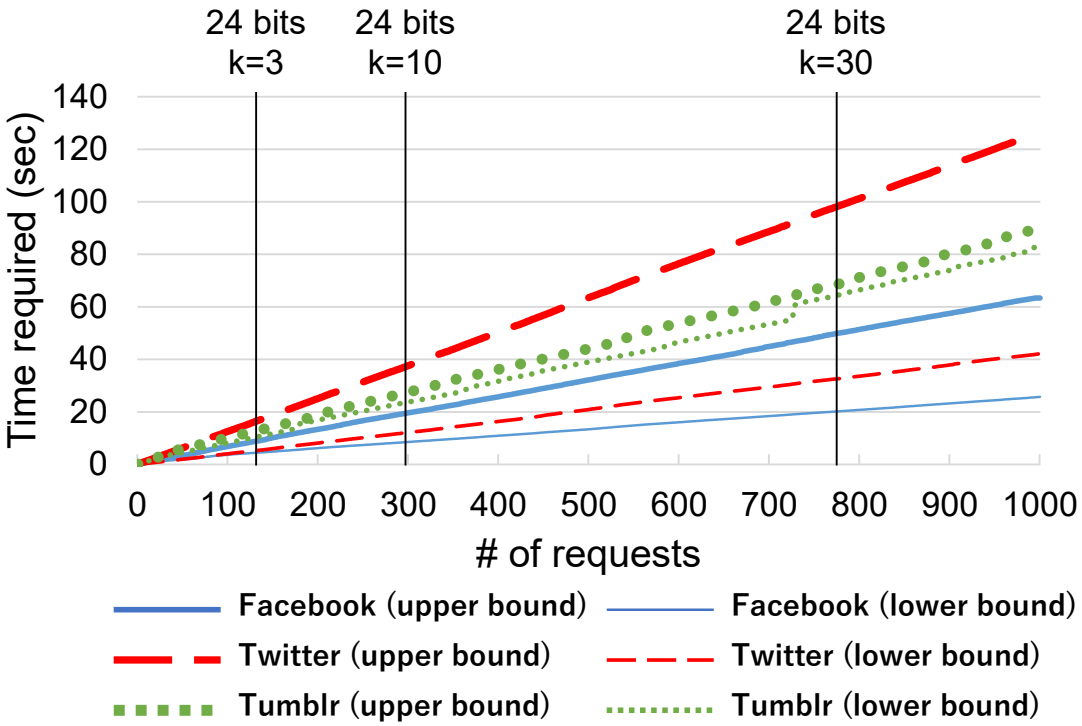


Fig. 3.7 Number of requests vs. time.

3.7.1 Principle of the Attack

We argue that the most fundamental assumption of our attack is the presence of the *visibility control* property in the system, that is, "given a multi-user web service, there exists a way for a (rogue) user to *control* what other users see, individually for each user". To be more formal, the part "what other users see" can be replaced with "any observable side-effect of the system caused by a certain action taken by a user". This assumption combined with a timing side-channel attack, which enables the attacker to steal this information from outside the system, is our attack's big picture. Because closing a side channel completely is well-known to be difficult, we believe that this visibility-control assumption is the main principle of this attack. In the case of our scenario, the ability to build signaling accounts using user blocking corresponds to this principle.

We stress that other services under this assumption, even those without user blocking but with a similar mechanism such as group invitations or file access permissions, may also be subject to a similar class of attack. Still, the social web/user-blocking example that we used in this chapter is by far the most practical application. This is likely because it satisfies several additional conditions: (1) the control can be done without the target's approval or notification and (2) the control can be done at a fine granularity, i.e., the different bits of information assignable per user is large. More specifically in our case, condition (1) is almost always achieved as an inherent nature of user blocking and condition (2) is achieved with unlimited granularity, in theory, via the creation of an arbitrary number of signaling accounts. Even though we omit further discussions concerning the presence of other such properties or the exploitability of similar systems, we believe that there is a need for further study concerning this subject.

3.7.2 Practical Aspects

Here we describe some of the applications and characteristics that extend and strengthen our attack from a practical perspective.

Target List Building.

With our attack, it is assumed that an attacker knows the identity of a victim account in advance, implying she/he has a list of identities for the target users. Thus, building a target list is a crucial factor for attack success. *A limited set of selected accounts* and *unspecified large number of accounts* are both attractive target candidates for an attacker. The former will include famous people (e.g., politicians, corporate officers, celebrities, YouTubers) or intimates of the attacker. In such cases, the target list can be built based on the attacker's knowledge or the existing open list [41]. The latter[*1] is realistic by using automation techniques, in particular, chasing the chain of friends' friends [42, 43]. Ugander et al. [44] revealed that

[*1] Even in the latter case, it is reasonable for the attacker to create subsets of all enumerated users according to user attributes (nationality, age, school, company) and reduce the number of user blocks. We discuss the limit of user blocking in Section 3.7.3.

99.91% of Facebook users belong to a single huge relationship graph, and Dey et al. [42] found that 82.73% of users publish their friend lists. Depending on the service feature, user searches and user groups may be useful for user enumeration. Some services, such as Twitter, can list all accounts simply by incrementing a numerical user identifier.

Identity Linking. User identification is only threatening if the identity is linked with another piece of information meaningful to the attacker. In the case of our attack, the most basic form of linking can take the form of linking the user's identity with *the fact that the user has visited the website* prepared for the attack. In this case, if the web content reflects the visitor's preference in any way, it may become a privacy concern. This is suitable not only for advertisement or access analyses, but also for various social engineering attacks or for blackmailing those who have accessed sites hosting pornographic content or illegal content such as pirated software. In addition, our attack can be implemented to reveal accounts on multiple services simultaneously and linking these accounts together could significantly worsen the impact of a privacy leak.

Another form of linking occurs when a person is induced to access the web server via an extra hop through another medium, resulting in a linking between the target's identity and the medium used. For example, on a social web service where the target's identity is already known, an attacker can send the target a message containing the URL of the web server. Note that this would allow the attacker to link even the web services which our attack cannot be applied to. Similarly, we can link non-web services, such as email or mobile text messages, which would result in linking an email address or phone number with a social account. Further, we can also link the target's *physical identity*, such as the target's physical presence or their residence, by placing or mailing a physical object, e.g., a poster or a flier, with URL, QR code or NFC tags printed on them. Note that, even though it may appear that revealing additional identities of a target when the target's other identities are already known is not so significant, it could lead to the identification of a target's anonymous account that cannot otherwise be discovered in a straightforward way.

Group Identification. Even though we have focused on the goal of *user* identifi-

cation in this chapter, we can easily extend this goal to *group* identification, that is, identifying not the user's exact identity but more general properties such as gender, nationality, or interests. The attacker could map each user to a bit array corresponding to the target attribute collected from the structured information available on the social web service. Note that this can be seen as a generalization of the user-space partitioning described previously, where a user space corresponds to a group of users with an arbitrary size mapped to a certain attribute. Group identification can be used by advertisement providers to track the visitor's attributes without unnecessarily revealing their user account. Note that the number of bits required for group identification would typically be much lower than that for user identification, making this attack significantly easier to execute than user identification.

Authentication-backed Identification. One major strength of our approach is that it is backed by the identity information guaranteed by the authentication system of the service, making it resilient against spoofing or misidentification, both of which many other methodologies suffer from. To give a simple example, when using an IP address for identification or even tracking, IP spoofing or ambiguity due to NAT or dynamic IP would interfere with this process. Note that social web service accounts are increasingly used as a building block in the modern web's authentication infrastructure. It is still possible to perform spoofing and one way is to create an account trying to mimic one's identity; however, scrutinizing the account content would usually easily reveal whether it is a spoofed account. Another way is to use a stolen account, but in this case, the victim user should be worried about much more serious problems than privacy leakage. In addition, because authentication is independent of the environment, it enables cross-environment (e.g., cross-device and cross-browser) identification and tracking, which is often difficult to achieve using other approaches.

3.7.3 Limitations

Login State Persistence. Our attack relies heavily on the assumption that the target user's service login state is alive while the user browses other websites. This as-

sumption is reliant on the web cookie mechanism; therefore, the cookie's expiration time or the user configuring the browser to clear cookies on closing the browser may affect the availability of our attack. Social web services, fortunately, tend to set a relatively long or even no expiration time, as seen in the commonly available "keep me logged in" features [45]. This is likely due to the incentives to service providers from a marketing perspective, e.g., tracking and advertisement, contrary to security-critical services such as Internet banking that set a short expiration time. In addition, users would lose the convenience of being able to access the service without the need to login every time, which may be a disappointing trade-off, especially for social web services which often assumes constant usage. Note that, simply determining whether a user is logged in to certain services can be accomplished in much more lightweight ways [46], which can also be used in our attack to pre-select the services to be targeted.

User Environment.

Our results from field experiments in Section 3.6 indicate that our attack is almost completely successful with various network environments and browser engines. However, we recognize that we could not consider all Internet user conditions. Especially, a non-negligible portion of users today access social web services from their mobile devices, so whether or not the attack is feasible in this realm is an important question to explore. For recent mobile platforms such as Android and iOS, the mechanics of most web browsers as well as the effective performance of the hardware and network are not significantly different from those of a PC; therefore, they are expected to yield sufficient RTT differences making our attack feasible. We partially proved this in our experiment with the tethering environment. The primary concern instead is the unique software ecosystem of mobile devices: many services encourage users to use a service-dedicated app instead of a browser to access their service. Even though some collaborative features such as social plug-ins or single sign-on may still urge some mobile users to log on via a browser, this ecosystem will surely limit the target coverage of our attack to a certain degree. We believe that a possible attack vector for this scenario which may need an attention might be an exploitation of a mobile platform-specific side channel, e.g., Android's Intent

and shared memory [47], to bypass the app sandbox, analogical to how our attack exploited a browser timing side channel to bypass the same-origin policy, but we leave further discussions on this for a future study.

Limits on Blocking. For most services, limitations on the total number of users allowed to be blocked or the rate at which blocking requests can be issued from a single account are not explicitly stated. We have experimentally confirmed that at least ten million users on Twitter and three million users on Facebook and Tumblr were actually blockable over five days using a single account, and only DeviantArt and eBay seems to have had a limit on the maximum number of blocks per account. Also, Instagram appears to have had a limitation on the rate, i.e, the number of accounts that can be blocked per minute. As we have shown in Section 3.2.2, neither disabling blocking nor posing a limit on it, is desirable from the viewpoints of the actual usage of the service and users' expectations. However, having limits on the total number of users to be blocked blocking may interfere with the process of building a high-coverage signaling account. Still, user-space partitioning would help alleviate this limitation and much of the effort for building signaling accounts is required just once, implying that attackers are not so exceedingly time-constrained when performing this task.

Length of Visit. As shown in Section 3.6, the attack can be executed in a realistically short time. In certain circumstances, however, such as when the RTT is high or when there is a need to use user-space partitioning, which increases the number of requests, it may be difficult to keep the user on the same webpage long enough for the JavaScipt code to finish. Even if the attack duration is short, because the behavior of a user is often unpredictable, a shorter attack is always preferable. A trivial approach to this problem is to prepare webpage content that is sufficiently "attractive" to cause the users to stay longer, but this is very user specific. Another solution is to save and restore the attack state between multiple attack sessions. By having the JavaScript code send partial results to the server as it attacks, even if the attack terminates before finishing, the attack can be resumed at another session from where it left off. Training data may be reused or not depending on the "distance" between each attack session, e.g., the time elapsed between sessions. Another solution is to open pop-up

windows in the background or a tab and execute the attack there, hoping that the user would not notice or care to close it immediately.

3.7.4 Research Ethics

In Section 3.2.2, all the data have been collected with user consent, and we followed guidelines presented by the ethics committee of Waseda University. To evaluate the feasibility and impact of the attack techniques on social web service users, experimenting with attacks on actual social web services cannot be avoided. All attacks in our experiment were checked manually and only generated a restricted amount of request. As a result, our experiment was carefully controlled and only generated a restricted amount of traffic (requests), which did not increase the workload of the sites and did not undermine the quality of their services. Furthermore, our experiment performed against our own accounts. Therefore, actual users of the services we examined were not directly involved in our attacks.

3.8 Defense

In this section, we discuss defensive measures that can be taken against our attack. We also present several countermeasures adopted by today's popular social web services. These countermeasures were developed with our aid.

3.8.1 Server-side Defenses

Access Validation. Token-based defenses are widely adopted to prevent CSRF attacks in general. The server appends a one-time random string, or token, to each URL link generated and verifies it when the link is accessed. A defense mechanism that validates a referrer is also effective because it can accept requests from the whitelisted URLs while rejecting all other illicit requests. These prevent any third-party from generating a valid link; therefore, the attacker will not be able to receive valid responses containing information useful for the attack as long as the access validation process is applied before the block checking at the server side. A major

drawback of these defenses is that legitimate requests are also affected and result in consequences such as breaking search engine results or prohibiting any means of link sharing, including those on blog posts and emails. Promising approaches which acquire user contents by using JavaScript's XMLHttpRequest with a valid token such as *placeholder* [38] and *double-submit cookie* [48] have been proposed, but they still require a change in the system architecture design and also the delay caused by the extra hop may negatively affect the user experience.

Response Time Control. The server could adjust the response time to minimize the block/non-block RTT difference. One approach is to artificially equalize the response times by adding delays to whichever has the shorter response time. Another approach is to randomize the response time by injecting delays of random lengths. However, either approach would impose a non-negligible performance degradation experienced by the user. In general, this type of timing side-channel defense is difficult to perfect; the profound study results in this area provide advanced attackers with various ways to amplify such differences at the cost of some increased effort, as we also have exemplified in this chapter. In addition, the network delay is often uncontrollable from the service side so a perfect control is difficult to attain from the server side. Note that such types of server-side defenses are often thwarted by other timing side-channel approaches, such as those leveraging the content cache [38].

Usage Restriction. Our attack, when implemented in a straightforward manner, may exhibit behavioral characteristics not usually seen in the normal usage of the service. One case of such an anomaly would occur in the preparation process of a signaling account, which requires a massive number of blocking requests to be issued within a short time. Another is in the process of launching the attack from a browser, which causes an abnormal number of GET requests to be issued. The service can either restrict this in the form of the rate limit, CAPTCHA, or some means of heuristic anomaly detection. However, these defenses are expected to function only as a mild mitigation, because advanced attackers have historically been able to circumvent these types of defenses. The most extreme form of restriction is to remove the user-blocking capability from the service. All these types of restriction-based measures, however, lead to an undermining of the ability to suppress those who truly needs to

be blocked, which may result in a degradation of the service quality.

3.8.2 Client-side Defenses

User. Defenses that can be taken by a user alone are limited to quite trivial ones. One approach is to isolate the browsing environment in which the web service is used, from that used for other purposes. This can be done, for example, by using the private browsing feature commonly available in modern browsers, logging out of the service when not in use, or simply using a different browser. Another approach is to restrict the execution of JavaScript using browser plug-ins such as NoScript [49], which would severely impair the attacker's capability to carry out such an attack. Obviously, all of these measures greatly increase the user's cost of not only using the service but also web browsing in general. Further, it would deactivate some features such as social plug-ins or advertisements that benefit both of the user and the service provider.

Web Browser.

SameSite is a cookie attribute that allows flexible control of sending cookies in cross-site requests. It is necessary for a browser to adopt this feature, and the web service explicitly declares it in the HTTP header. Since browsers behave as if a user is not logging in when they make cross-site requests from a third-party site to a social web service, the difference in RTTs between blocking and non-blocking states disappears in principle. A Web service that adopts *SameSite* has two options: `samesite=strict` and `samesite=lax`. For the `strict` option, browsers remove cookies from any cross-site request, including redirects and link clicks. As with access validation, legitimate page transitions may be disturbed as well. For the `lax` option, however, browsers do not prevent the sending of cookies for cross-site requests with top-level navigation. Thus, the `lax` option can be used to defend against various CSRFs including timing attacks while not sacrificing user experience. However, `samesite=lax` still interferes with the functionality of some social plug-ins that leverage cookies from the social web such as Facebook Comments.

Equalizing the response times, for example, by injecting delays to the processing

time, is also a possible measure that can be taken on the browser side. Further, the detection of anomalies such as frequent errors resulting from failed rendering may be another option. However, these approaches are often only viable for a certain class of timing side channels; they tend to be thwarted eventually by other newly developed timing attacks using different approaches, as exemplified by the attack using the browser cache mentioned in another study [38]. Cao et al. [50] proposed *Deterministic Browser*, which prevents JavaScript program from accessing the physical clock when a secret event is running at the same time. Instead, they provide an auxiliary clock that indicates virtual time to make timing attacks impossible. This approach works against our attack, but it can hinder the browser rendering for some websites.

3.8.3 Responsible Disclosure

Even though the attack technique in this chapter does not arise from a specific social web service, according to the principle of responsible disclosure, we have reported the details of our attacks and the experimental results to the relevant social web service providers and browser vendors to mitigate the attacks and improve future security design of social web.

Through these activities, Twitter has been able to prevent the threat of account identification by changing their specifications [51] to improve their security mechanism. Their primary defense approach is using the *SameSite* attribute. This is currently working for most users, as we and Twitter encourage major browser vendors (including Microsoft and Mozilla) to adopt *SameSite*. For older browser versions, Twitter's other mitigation still prevents our attack. It is referrer-based access validation, but a user with invalid referrers is not simply denied. Instead, such a user is redirected to a stepping-stone page with no difference in RTT with only minimal JavaScript code then redirected again to the destination page via JavaScript. This works as a defense because the time taken to redirect via JavaScript cannot be measured under the constraints of the same-origin policy. While this approach slightly increases the number of requests, users coming from external links reach the desired content on the social web. However, the user cannot use browsers (or

browser extensions) that remove the referrer for privacy.

Although we refrain from disclosing the brand names, several other service providers and browser vendors have already finished implementing defenses. Some services have decided not to adopt countermeasures. They considered the cost required for implementing defenses and the degradation of user experience due to specification changes. We leave investigating these trade-offs between the feasibility of defenses and the risk of user privacy for our future challenge.

3.9 Related Work

We present previous studies concerning timing attacks, which is the fundamental technique of our method uses to compromise user's privacy. In addition, we introduce other side-channel leaks based on the browser functionality and methods to identify and track users.

3.9.1 Web-based Timing Attacks

A timing attack is one type of side-channel attack that has been studied primarily in cryptography for more than two decades. It typically exploits the execution time or power consumption of a cryptosystem to infer secret key and private information [52, 53]. Studies of timing attacks have expanded to web-based systems regardless of the cryptosystem that exploits the communication time and size of the web content. Bortz et al. presented a pioneer work on web-based timing attacks; they classified web-based timing attacks into *direct timing* and *cross-site timing* [35]. Our proposed method is classified as a web-based cross-site timing attack.

A direct timing attack directly measures the response times from a system, e.g., a website, to extract private information from a system. Bortz et al. proposed a method to expose valid user names and the number of private photos from a website by measuring the response time of HTTP [35].

Cross-site timing attacks indirectly measure the response times or content size of web on a browser to extract private information from a browser or website. It enables a malicious website to obtain information about the target browser's view of another

website using cross-site content that often violates the same-origin policy [54]. Methods to break the same-origin policy and their countermeasures have been presented since 2000 [55–58]; however, the many of cross-origin techniques are still effective on modern web browsers. Liang et al. leveraged several CSS features to indirectly monitor the rendering of a target resource [59]. Goethem et al. proposed a cache-based timing attack using HTML5 functionalities, which can bypass the same-origin policy, to estimate the size of a cross-origin resource [38]. Gelernter et al. presented a *cross-site search attack* on well-known web services to distinguish between the loading time of empty and full responses, which enables an attacker to distinguish sensitive data of target users in the records of the web services [60]. Jia et al. demonstrated a *geo-location inference attack* on well-known web services, by using the load time of location-sensitive resources left by geography-specific websites (e.g., Google's local domain) [61]. Our method is not new in the context of cross-site attacks; however, the idea is unique in that user blocking, which is a fundamental functionality of social webs, can be used to distinguish between the blocked and non-blocked states, consequently, to identify their social accounts.

3.9.2 Side-channel Leaks on Browsers

A side-channel attack on a browser without timing features is another class of privacy attack. To infer the status of a cross-origin resource, Lee et al. developed a *URL status identification attack* using `ApplicationCache` that exploits cross-origin resource caching [46] and they suggested advanced privacy threats using this attack, e.g., login status determination and internal web server probing. A *history-stealing attack* is a typical attack that extracts the browsing history of URLs [39, 62]. This attack depends on the fact that a web browser handles CSS properties of URL hyperlinks differently depending on whether the URL was previously accessed by the web browser [63], which leads to allowing a client-side script to access such properties. To fix this, Baron proposed a solution that blocks scripts from accessing the CSS properties of hyperlinks, and all popular browsers (e.g., Firefox, Chrome, Safari, and IE) have adopted this solution. As a result, this type of history stealing attack no longer works in the latest versions of these browsers [64, 65].

3.9.3 Social Account Identification

While various methods have been proposed to effectively track browsers on the Internet (e.g., cookies, browser cache, and browser fingerprints [66–68]), these tracking methods focus on identifying distinct browsers rather than the user of the browsers. The goal of our proposed method is to identify the user (i.e., the social account) which differs from the above browser tracking methods. Many of the studies introduced in Sections 3.9.1 and 3.9.2 mentioned that their proposed methods could be used for inferring the status of social account or identifying social account [35, 39, 46]. The difference of response time of login page was used for inferring account validity [35]. With a similar motivation, conditional redirections of the HTTP URLs was used for distinguishing whether a victim web browser is logged in to the web service [46]. The combination of *group membership information*, e.g., group ID or group directory in browser's access history, was used for identifying a social account [39]. These differences are extracted from previously provided pages, e.g., login pages and group membership pages. In contrast, our method is unique in that an attacker can fully control the visibility of pages in order to create discriminable differences.

3.10 Conclusion

This chapter presents a practical side-channel attack that identifies the social account of a user visiting the attacker's website. It exploits the user-blocking mechanism, or the visibility control property, commonly available in most social web services today to create a controllable side channel that provides the attacker with complete and flexible control over the leaked information, be it informative enough to uniquely identify the user or be it highly resilient to noise. With experiments, we demonstrated that our attack is in fact applicable to current mainstream social web services today and we argued that defending against this threat would not be easy without imposing a negative impact on the relevant services. It is ironic that the blocking feature designed to suppress harmful users can now be turned against harmless users; some

form of mitigation is urgent and a reworking of the design of this feature is suggested and major services and browsers adopted new security features.

Chapter 4

Analyzing the Inconsistency between Behaviors and Descriptions of Mobile Apps

4.1 Introduction

Most applications for mobile devices are distributed through mobile software distribution platforms that are usually operated by the mobile operating system vendors, e.g., Google Play and Apple App Store. Third-party marketplaces also attract mobile device users, offering additional features such as localization. According to a recent report published by Statista [69], the number of mobile app store downloads in 2017 are expected to exceed 197 billion. Mobile software distribution platforms are the biggest distributors of mobile apps and should play a key role in securing mobile users from threats, such as spyware, malware, and phishing scams.

As many previous studies have reported, privacy threats related to mobile apps are becoming increasingly serious, and need to be addressed [70–73]. Some mobile apps, which are not necessarily malware, can gather privacy-sensitive information, such as contact list [74] or user location [75]. To protect users from such privacy threats, many of mobile app platforms offer mechanisms such as permission warnings

and privacy policies. However, in practice, these information channels have not been fully effective in attracting user attention. For instance, Felt et al. revealed that only 17% of smartphone users paid attention to permissions during installation [72]. The Future of Privacy Forum revealed that only 48% of free apps and 32% of paid apps provide in-app access to a privacy policy [76]. Further more, Chin et al. reported that roughly 70-80% of end users ignored privacy policies during installation process [77].

Let us turn our attention to a promising way of communicating with users about apps and privacy. This information channel is the *text descriptions* provided for each app in a marketplace. The text description is usually written in natural, user-friendly language that is aimed to attract users' attention; it is more easily understood than the typical privacy policy. In addition, when a user searches for an app in a marketplace, s/he create query keywords, which are generally searched on text descriptions. Then, users review the search results, often by reading the text descriptions; i.e., text descriptions can work as a proxy to the user expectations. In fact, text descriptions have a higher presence than permission warnings or privacy policies, and therefore, are a good channel for informing users about how individual apps gather and use privacy-sensitive information.

With these observations in mind, this work aims to address the following research question through the analysis of huge volume of Android applications:

> **RQ**: *What are the primary reasons that text descriptions of mobile apps fail to refer to the use of privacy-sensitive resources*?

The answers to the question will be useful for identifying sources of problems that need to be fixed. To address the research question, we developed a framework called ACODE (Analyzing COde and DEscription), which combines two technical approaches: static code analysis and text analysis. Using the ACODE framework, we aim to identifiy reasons for the absence of the text descriptions for a given privacy-sensitive permission. Unlike the previous studies, which also focused on analyzing the text descriptions of mobile apps [78–81], our work aims to tackle with a huge volume of applications. To this end, we adopt light-weight approaches, static

code analysis and keyword-based text analysis as described below.

Our static code analysis checks whether a given permission is declared. Then, it investigates whether the code includes APIs or content provider URIs [*1] that require permission for accessing privacy-sensitive resources. Lastly, it traces function calls to check that the APIs and/or URIs are actually *callable* to distinguish them from apps with dead APIs/URIs that will never be used; e.g., reused code could include chunks of *unused* code, in which privacy-sensitive APIs were used.

Our description analysis leverages techniques developed in the fields of information retrieval (IR) and natural language processing (NLP) to automatically classify apps into two primary categories: apps with text descriptions that refer to privacy-sensitive resources, and apps without such descriptions. Here we present three noteworthy features of our approach. First, since we adopt a simple keyword-based approach, which is language-independent, we expect that it is straightforward to apply our text analysis method to other spoken languages. In fact, our evaluation through the multilingual datasets demonstrated that it worked for both languages, English and Chinese. Second, although our approach is simple, it achieves a high accuracy for nine distinct data sets. The accuracy is comparable to the existing pioneering work, WHYPER [79], which makes use of the state-of-the-art NLP techniques. The reason we developed the ACODE framework instead of using the WHYPER framework was that we intended to extend our analysis to multiple natural languages. The WHYPER framework leverages API documents to infer semantics. As of today, Android API documents are *not* provided in Chinese. Accordingly, we were not able to make use of the WHYPER framework to analyze Chinese text descriptions. Finally, like the WHYPER framework, our text analysis technique does *not* require manually labeled descriptions. Therefore, it enables us to enhance the text analysis of descriptions to any permission APIs without requiring expensive labeling tasks. It also enables us to reduce cost of text analysis significantly. The key idea behind our approach is to leverage the results of code analysis as a useful

[*1] Content providers manage access to data resource with permission using Uniform Source Identifiers (URIs); for instance, `android.provider.ContactsContract.Contacts.CONTENT_URI` is an URI used to get all users registered in the contact list.

hint to classify text descriptions.

To the best of our knowledge, only a few previous studies have focused on analyzing the text descriptions of mobile apps [78–80]. A detailed technical comparison between these studies and ours is given in section 4.7 (see Table 4.10 for a quick summary), and here we note that this work is distinguishable from other studies by being an extensive empirical study. The volume of our dataset is several orders of magnitude larger than previous studies. In addition, because we wanted to extract generic findings, we conducted our experiments in such a way as to incorporate differences in the resources accessed, market, and natural language. Our analysis considered access of 11 different resources taken from 4 categories, i.e., personal data, SMS, hardware resources, and system resources (see Table 4.1). We chose the resources because they are the most commonly abused or the potentially dangerous ones. We collected 100,000 apps from Google Play and a further 100,000 apps from third-party marketplaces. We also collected 10,000 paid apps from Google Play for comparison. For the natural language analysis, we adopted English and Chinese, because they are the two most widely-spoken languages worldwide [82]. Furthermore, to evaluate the performance of text analysis, we obtained a total of $6,000$ text descriptions from 12 participants. Each description was labeled by three distinct participants.

The key findings we derived through our extensive analysis are as follows:

The primary factors that are associated with the inconsistencies between text descriptions and use of privacy-sensitive resources are broadly classified into the following four categories.:

(1) **App building services/frameworks**: *Apps developed with cloud-based app building services or app building framework, which could unnecessarily install many permissions, are less likely to have descriptions that refer to the installed permissions.*

(2) **Prolific developers**: *There are a few prolific developers who publish a large number of applications that unnecessarily install permissions and code.*

(3) **Secondary functions**: *There are some specific secondary functions that require access to a permission, but tend to be unmentioned; e.g., 2D barcode*

reader (camera resource), game score sharing (contact list), and map apps that directly turns on GPS (write setting), etc.

(4) **Third-party libraries**: *There are some third-party libraries that requires access to privacy-sensitive resources; e.g., task information (crash analysis) and location (ad-library, access analysis).*

The main contribution of our work is the derivation of these answers through the extensive analysis of huge volume of datasets. We believe that these findings will be useful for identifying sources of problems that need to be fixed to improve the users' awareness of privacy on mobile software distribution platforms. For instance, as our analysis revealed, there are several HTML5-based app-building framework services that unnecessarily install permissions, which could render the system vulnerable to additional threats of malicious JavaScript injection attacks. Therefore, an app developer should not install unnecessary permissions. However, if a developer used a rogue app-building framework service, he/she may likely not be aware of unnecessary permissions installed. ACODE enables operators of mobile software distribution platforms to pay attentions to these cases, which are invisible otherwise.

The rest of this chapter is organized as follows. Section 4.2 describes our the ACODE framework in detail. In section 4.3, we show the details of the static code analyzer. Section 4.4 contains details of the text description classifier. We present our findings in section 4.5. Section 4.6 discusses the limitations of ACODE and future research directions. Section 4.7 summarizes the related work. We conclude our work in section 4.8.

4.2 ACODE framework

In this section, we provide an overview of the ACODE framework. We also connect the components of the ACODE framework to the corresponding sections where we will give their details.

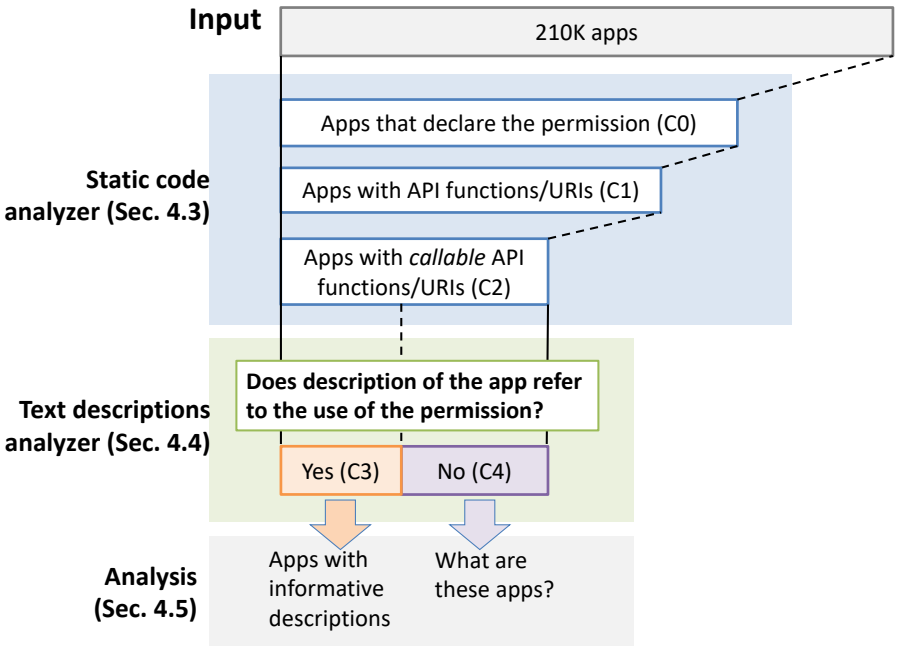


Fig. 4.1 Overview of the ACODE framework.

4.2.1 Goal and overview

Figure 4.1 is an overview of the ACODE framework. As discussed previously, we used a two-stage filter, employing a static code analyzer and text descriptions analyzer. In the first stage, the first filter extracted apps that declare at least one permission, e.g., location (C0). The second filter extracted apps with code that include corresponding APIs/URIs (C1). The third filter checked whether the APIs/URIs are *callable* from the apps by employing function call analysis (C2). In the second stage, the text classifier determined whether the text descriptions refer to the use of location explicitly or implicitly (C3), or not at all (C4). Note that we are not considering apps that do not declare to use permission, but have descriptions that indicate that permission is needed.

These filtration mechanisms enabled us to quantify the effectiveness of text descriptions as a potential source of information about the use of privacy-sensitive

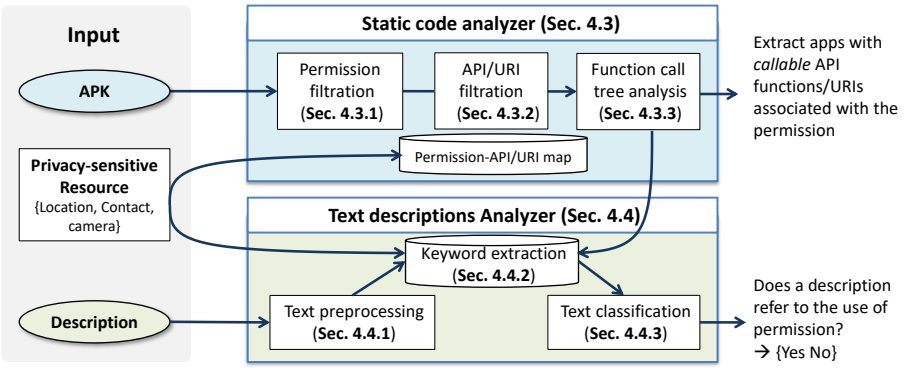


Fig. 4.2 Components of the ACODE framework.

resources. For instance, by counting the fraction of apps that are classified as C3 (see figure 4.1), we can quantify the fractions of apps with text descriptions that successfully inform users about the use of privacy-sensitive resources for each resource. By examining the sources of apps that are classified as C4, we can answer our research question, **RQ**. The detailed analysis will be shown in section 4.5.

Figure 4.2 illustrates the components used in the ACODE framework. For each application, we had an application package file (APK) and a description. APK is a format used to install Android application software. It contains code, a manifest file, resources, assets, and certificates. The text descriptions of apps were collected from mobile software distribution platforms. As shown in the figure, the APKs and text descriptions were input to the static code analyzer and description classifier, respectively.

4.2.2 Static code analyzer

The goal of the static code analyzer is to extract APK files whose code include *callable* APIs/URIs that are required to use permissions related to a privacy-sensitive resource. For a given permission, first, we extracted apps that declare the use (C1, see section 4.3.1). Then, we checked whether disassembled code of the app include the APIs/URIs, which require the permission (C2, see section 4.3.2). If code included at least one API or URI, then, we checked whether it was actually *callable* within

the app by investigating the function call graph with some heuristics we developed (C3, see section 4.3.3). It should be noted that the static code analysis has some limitations that we will discuss in section 4.6.

4.2.3 Description classifier

The goal of the description classifier was to classify text descriptions into two categories: those that refer to the use of a resource (C3), and those that do not (C4). In other words, we wanted to determine automatically whether a user can, by reading the text description, know that an app may use a privacy-sensitive resource. To do this, we leveraged several text analysis techniques. We also make use of the results of code analyzer to extract keywords associated with a resource. To extract keywords that are useful in classifying text descriptions, we first present text data preprocessing techniques in section 4.4.1. Next, in section 4.4.2, we present the keyword extraction method that leverages techniques used in the field of information retrieval. We also evaluate the accuracy of the description classifier in section 4.4.3.

4.3 Static Code Analysis

This section describes the static code analysis techniques used in the ACODE framework. The purpose of static code analysis was to extract apps that include *callable* APIs/URIs to use a given permission. Before applying function call analysis, which is a process of checking whether given function is callable, we applied two filtration mechanisms: (1) permission filtration and (2) API/URI filtration. These filtrations are effective in reducing the computation overhead needed for function analysis. We also note that permission filter is useful to prune apps that include callable APIs/URIs, but will not actually use it.

4.3.1 Permission filtration

First, we applied permission filtration, which simply checks whether an app declares a given permission. According to Zhou et al. [83], permission filtration is quite

Table 4.1 List of permissions used for this work.

Category	Permission	Definition*
Personal data	ACCESS_FINE_ LOCATION	Allows an app to access precise location from location sources such as GPS, cell towers, and Wi-Fi.
	GET_ACCOUNTS	Allows access to the list of accounts in the Accounts Service.
	READ_CONTACTS	Allows an application to read the user's contacts data.
	READ_CALENDAR	Allows an application to read the user's calendar data.
SMS	READ_SMS	Allows an application to read SMS messages.
	SEND_SMS	Allows an application to send SMS messages.
Hardware resources	CAMERA	Required to be able to access the camera device.
	RECORD_AUDIO	Allows an application to record audio.
System resources	GET_TASKS	Allows access to the list of accounts in the Accounts Service . (This constant was deprecated in API level 21)
	KILL_BACKGROUND_ PROCESSES	Allows an application to call killBackgroundProcesses(String).
	WRITE_SETTINGS	Allows an application to read or write the system settings.

*http://developer.android.com/reference/android/Manifest.permission.html

effective in reducing the overhead of analyzing a huge amount of mobile apps. For each app, we investigated its AndroidManifest.xml file to check whether it declares permissions to access given resources. The process can be easily automated using existing tools such as `aapt` [84]. To further accelerate the data processing, we also leveraged multiprocessing techniques. Table 4.1 summarizes the 11 different permissions we analyzed in this work. To perform generic analysis, we chose the permissions from 4 categories, personal data, SMS, hardware resources, and system resources. These resources were chosen because they are the most commonly abused or the potentially dangerous ones.

4.3.2 API/URI filtration

Next, for each sample, we checked whether it includes APIs or content provider URIs that require permissions to access privacy-sensitive resources. For this task, we made use of the API calls for permission mappings extracted by a tool called PScout [85], which was developed by Au et al. [86]. In addition to API-permission mapping, the PScout database also includes URI-permission mapping. To check the existence of APIs or URIs, first, using Android apktool [87], we extracted DEX code

from APK files and disassembled them into smali format [88]. Then, we checked whether a set of APIs is included in the code of an APK file.

We note that some apps may require permissions but not include any APIs or URIs that request the permission. This may occur for several reasons. apps. If such possibly overprivileged apps are simply overprivileged due to developer's error, they do not impact our study, because those apps may not need to use APIs or URIs. However, as Felt et al. [71] reported, one of the common developer errors that cause overprivilege is Intent. A sender application can send an Intent to a receiver application, which uses permission API. In such cases, the sender of the Intent does not need to have permissions for the API. We saw many such cases, especially related to camera permissions. In fact, [71] reported that of the apps that unnecessarily request camera permission, 81% send an Intent to open the already installed camera applications (including the default camera) to take a picture. Our observation is in agreement with their finding.

Thus, our API/URI filtration scheme may miss a non-negligible number of apps that actually use the camera through Intent. However, note that our final analysis will be applied to the apps in set $C2$ as shown in figure 4.1. Therefore, we are confident that the removal of such apps should not affect our analysis, because we do not expect to see significant differences between the descriptions of those apps removed due to the Intent problem and the descriptions of apps included in $C2$.

4.3.3 Function call analysis

Now, we present the function call analysis of the ACODE framework. For convenience sake, let the term function include method, constructor execution, and field initialization; i.e., we trace not only method calls, but also class initializations. Figure 4.3 presents a pseudo-code of the algorithm we developed for function call analysis. It checks whether APIs/URIs of a given permission are callable (true) or not (false). The algorithm uses depth-first search to search the function call tree. If it finds a path from the given function to a class of `ORIGIN` (line 4), it concludes that the app has at least one API/URI that is callable, where `ORIGIN` is composed of three classes: `Application`, `App Components`, and `Layout`. `Application` is

```
1:  INPUT
2:  p : a permission
3:  a : an application (APK)
4:  ORIGIN =  [Application, App Components, Layout]
5:  list = getAU(p,a)  # list of APIs/URIs associated with p
6:  done = []          # empty list
7:
8:  WHILE list is not empty DO
9:      f = list.pop()
10:     IF f is in done:
11:         skip the function
12:     ENDIF
13:     IF f.parentClass is in ORIGIN:
14:         RETURN True
15:     ENDIF
16:     IF (f.parentClass inherits Android SDK)
17:         AND (f is not init)
18:         AND (f is not a static method):
19:         list.append(f.parentClass.init)
20:     ELSE IF (f is referenced):
21:         list.append(f.refFunctions)
22:     ENDIF
23:     done.append(f)
24:  ENDWHILE
25:  RETURN False
```

Fig. 4.3 Pseudo-code that checks the callability of APIs of a permission.

a class that initiates an Android app. It is called when an app is launched. `App Components` are the essential building blocks that define the overall behavior of an Android app, including `Activities`, `Services`, `Content providers`, and `Broadcast receivers`. While the `Application` and `App Components` classes need to be specified in the manifest file of an app, the `Layout` class does not. It is often used by ad libraries to incorporate ads using XML.

`getAU` (Line 5) is a function that returns a list of APIs/URIs for a given permission. As an implementation of getAU, we adopted PScout [85]. `refFunctions` (line 21) is a function that returns a list of functions that reference to the given function or URI. As an implementation of `refFunctions`, we adopted androguard [89], which we modified to handle URIs. If a function of a class, say Foo, implements a function of the Android SDK class whose code is not included in the APK, we cannot trace the path from the function in some cases. To deal with such cases, we made a heuristic to trace the function that calls the init-method of class Foo (lines 16–19). We note that the heuristics can handle several cases such as async tasks, OS message handlers, or callbacks from framework APIs such as `onClick()`. A method is callable if it is overridden in a subclass or an implementation of the Android SDK and an instance of the class is created. Async tasks, the OS message handler, or other callbacks implement their function by overriding the methods of the Android SDK subclass. Therefore, it should be handled by the heuristics.

4.4 Text Description Analysis

This section describes the text description analysis used in the ACODE framework. The aim of this analysis was to classify descriptions into two classes: (1) text descriptions that reference a privacy-sensitive resource, and (2) text descriptions that do not. To this end, we adopted a set of basic techniques used in both IR and NLP fields. As we shall see shortly, our keyword-based approach is quite simple and works accurately for our task. As Pandita et al. [79] reported, a keyword-based approach could result in poor performance if it was designed naively. So, we carefully constructed our keyword extraction processes. As a result, we achieved

87-98% of accuracy for the combinations of 3 resources and two languages. Simple and successful text description classification enabled us to automate the analysis of more than 200,000 text descriptions.

Section 4.4.1 describes how we preprocessed the description data so that we can extract keywords that are useful in classifying text descriptions. Section 4.4.2 presents the keyword extraction method that leverages techniques used in the field of information retrieval. Section 4.4.3 describes our experiments to compare our description classifier with the WHYPER framework in terms of accuracy.

4.4.1 Text Data Preprocessing

To analyze natural language text descriptions, we applied several text preprocessing methods. These methods are broadly classified into four tasks; (1) generic text processing, (2) domain-specific stop words removal, (3) feature vector creation, and (4) deduplication. Especially the tasks (2) and (4) are crucial in extracting good keywords that can accurately classify the text descriptions.

Generic text preprocessing

We first apply widely-used generic text preprocessing techniques: word segmentation, stemming, and generic stop words removal. Word segmentation is a process of dividing text into words. This process is required for Chinese but not for English, in which words are already segmented with spaces. We used KyTea [90] for this task. For English, we applied stemming, which is a process of reducing derived words to their stem. It is known to improve the performance of text classification tasks. We used NLTK [91] for this task. Note that the concept of stemming is not applicable to Chinese. Lastly, we applied generic stop words removal, which is a process of removing a group of words that are thought to be useless for classification tasks because they are commonly used in any documentation (e.g., determiners and prepositions). As lists of stop words, we used the data in NLTK [91] for English and the data in imdict [92] for Chinese.

Domain-specific stop words removal

Next, we created domain-specific stop words list so that we can remove terms that are not generic stop words but are commonly used in mobile app descriptions; e.g., "app" or "free". To this end, we make use of the technique proposed in Ref. [93], which is a term-based sampling approach based on the Kullback-Leibler divergence measure. Since the technique measures how informative a term is, we can remove the least weighted terms as the stop words. Number of sampling trial was set to 10,000. When we changed the threshold of extracting the top-L stop words; i.e., from $L = 20$ to $L = 150$, the following results are not affected at all. In the followings, we use $L = 100$. The extracted domain-specific stop words for English include "app", "free", "get", "feature", "android", "like", etc.

Feature vector creation

Using the preprocessed descriptions, we created a binary feature vector for each text description as follows. Let $\mathbf{W} = \{w_1, w_2, ..., w_m\}$ be a set of entire words after the screening process shown above. A feature of vector of the ith text description is denoted as $\mathbf{x}_i = \{x_i(w_1), x_i(w_2), ..., x_i(w_m)\}$, where $x_i(w_j) = 1$ if w_j is present in the ith text description. If w_j is not present, $x_i(w_j) = 0$.

Deduplication

Because we adopt the keyword extraction approach based on *relevance weights* as shown in the next subsection, the deduplication process plays a crucial role in eliminating the effect of same or similar descriptions generated by a single developer. For instance, if a developer produces thousands of apps with the same text description, which is often the case we observe in our datasets, the words included in the apps may cause unintended biases when computing the relevance weights of terms. To deduplicate the descriptions, we remove the same or similar descriptions by using the cosine similarity measure; i.e., for a given pair of feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, the cosine similarity is computed as $s = \cos\left(\mathbf{x}_i \cdot \mathbf{x}_j / |\mathbf{x}_i||\mathbf{x}_j|\right)$, and if s is larger than a threshold, the duplicated description is removed. We note that the value of threshold was not sensitive to the succeeding keyword extraction results if it is set between 0.5

to 0.8.

4.4.2 Keyword Extraction

To extract keywords, we leverage the idea of *relevance weights*, which measures the relation between the relevant and non-relevant document distributions for a term modulated by its frequency [94]. Relevance weighting was developed in the IR community as a means to produce optimal information retrieval queries. To make use of the relevance weights for our problem, we need to have sets of relevant and non-relevant documents. Since we do not have any labels that indicate whether a document is relevant, i.e., it refers to a permission, or non-relevant, i.e., it does not refer to a permission, we set the following assumption.

Assumption: *For a given permission, descriptions of apps that declare the permission and have callable APIs can be regarded as "pseudo relevant document", while the descriptions of the remaining apps can be regarded as "pseudo non-relevant document".*

Note that our research question contradicts with this assumption; i.e., we are interested in the reason why an app with callable API for a permission does not refer to the permission. Nevertheless, our performance analysis using multiple permissions in two spoken languages empirically supports that our approach actually works well in extracting effective keywords.

Under this assumption, we calculate the relevance weights for each word as follows. For a word w_i, the relevance weight (RW) is

$$RW(w_i) = \log \frac{(r_i + 0.5)(N - n_i - R + r_i + 0.5)}{(n_i - r_i + 0.5)(R - r_i + 0.5)},$$

where r_i is the number of relevant documents word w_i occurs in, R is the number of relevant documents, n_i is the number of documents word w_i occurs in, and N is the number of documents, respectively.

Using the entire descriptions with code analysis outputs, we extracted the keywords that have the largest relevance weights. Table 4.2 presents a subset of extracted keywords for each permission. For space limitation, we present only the Top-3 English keywords. In most cases, the keywords look intuitively reasonable.

Table 4.2 Extracted top-3 keywords for English descriptions.

Resources	1st	2nd	3rd
Location	gps	location	map
Account	grab	google	youtube
Contact	sms	call	contact
Calendar	calendar	reminder	meeting
SMS (read)	sms	message	incoming
SMS (send)	sms	message	sent
Camera	camera	scan	photo
Audio	recording	voice	record
Get tasks	lock	security	task
Kill background process	task	kill	manager
Write setting	alarm	ring	bluetooth

Interestingly, some keywords such as "sms" are found in multiple resources; i.e., contact, SMS (read), and SMS (send). In fact, these resources tend to co-occur. In the following, we will use these keywords to classify descriptions. Once we compiled the keywords, the text classification task is straightforward. If a text description includes one of the extracted keywords for a permission, the description is classified as positive, i.e., it refers to the permission. The problem is how we set the number of keywords to be used. We will study the sensitivity of the threshold in Section 4.4.3.

4.4.3 Performance Evaluation

To evaluate the accuracy of our scheme, we use manually labeled data sets. We first present the way how we compile the labeled data set. Next, we evaluate the accuracy of our approach, using the labeled data. Finally, to validate the robustness of our approach, we use the external dataset and compare the performance with the existing state-of-the-art solution, the WHYPER framework. In the analysis of accuracy (Section 4.4.3), we use 200,000 free apps, which will be described in Section 4.5.1 as *training sets*; i.e., they are only used for keyword extraction. The *labeled test set* is a subset of those, on which we measure accuracy. We note that in the evaluation, our training set included test set; i.e., we extracted the keywords

using the entire text descriptions, which is the training set, and applied the keywords (i.e., classifier) to the labeled descriptions, which is the test set. In general, training classifier using test set is not good because such setting could over-estimate the accuracy of the model. However, the effect should be small because our classifier was based on frequencies of terms and the test set accounted for only 0.6% of entire samples.

Creation of labeled datasets

We created the labeled data sets with the aid of 12 international participants who are from China, Korea, Thailand, and Indonesia. All the participants were university students with different disciplines in science and engineering. 7 were female and 5 were male. 4 were native English speakers, and 8 were native Chinese speakers. None of them had experience of developing Android applications. All the native Chinese speakers were fluent in English (native level). Students who were native speakers of Chinese labeled Chinese descriptions. In summary, six students labeled English descriptions, and the other six labeled Chinese descriptions. Here, we picked up three distinct resources, i.e., location, contact, and camera, out of the 11 resources we considered in this work.

Since a resource is used for various purposes, and referred to by various terms, we wanted to avoid participants focusing too much on a particular keyword, such as "camera". Instead, we asked participants to identify whether an app will use a camera, rather than whether it mentions a camera. This enabled us to identify several interesting keywords, such as "QR" and "scan". Also, we note that the question should reflect users' *awareness* of a resource.

Before asking participants to label text descriptions, we picked some descriptions from our entire data set. If random sampling were applied to the entire set, there would be a significant imbalance between the two classes. In particular, there would be very few positive samples, i.e., text descriptions that reference a resource. To avoid such an imbalance, we applied the access permission filter shown in section 4.3.1 so that the sampled text descriptions would include a certain number of positive samples. Although this solution could create some bias toward the positive class,

Table 4.3 Summary of labeled datasets.

English			
	Location	Contact	Camera
# of descriptions	1,000	1,000	1,000
# of labels	3,000	3,000	3,000
Chinese			
	Location	Contact	Camera
# of descriptions	1,000	1,000	1,000
# of labels	3,000	3,000	3,000

in fact it did not matter, as will be shown later in this chapter. From the set of apps that declare access permissions for using resources, we randomly sampled 1,000 text descriptions. In total, we sampled 6,000 descriptions, as shown in table 4.3.

Having sampled text descriptions, we asked each participant to label 500 text descriptions for each resource (e.g., $500 \times 3 = 1,500$ descriptions in total). A participant labeled text descriptions in either English or Chinese. To increase the quality of labels, each text description was labeled by three distinct, fixed participants. We obtained a total of $18,000$ labels for $6,000$ text descriptions, as shown in table 4.3.

Finally, we eliminate inconsistent labels to ensure that the quality of labels is high; i.e., we used only the text descriptions upon which all three evaluators agreed. Table 4.4 summarizes the text descriptions that met this criterion. We used these labeled descriptions for evaluating accuracy of our approach, as described in the next subsection.

Threshold Sensitivity Study

Using the labeled datasets, we empirically studied the relation between threshold and classification accuracy. Here, the definition of the accuracy is the fraction of correctly classified text descriptions, using the top-K keywords. Figure 4.4 presents how the number of keywords, K is correlated with the classification accuracy. As shown in the graph, across the 6 of labeled datasets, the accuracy is fairly stable around $K = 3$. Also, we notice that $K = 3$ gives the highest accuracy with

Table 4.4 Statistics of labeled descriptions to be used for performance evaluation.

English			
	Location	Contact	Camera
# of positive descriptions	128	208	276
# of negative descriptions	611	449	289
Chinese			
	Location	Contact	Camera
# of positive descriptions	38	102	157
# of negative descriptions	828	544	583

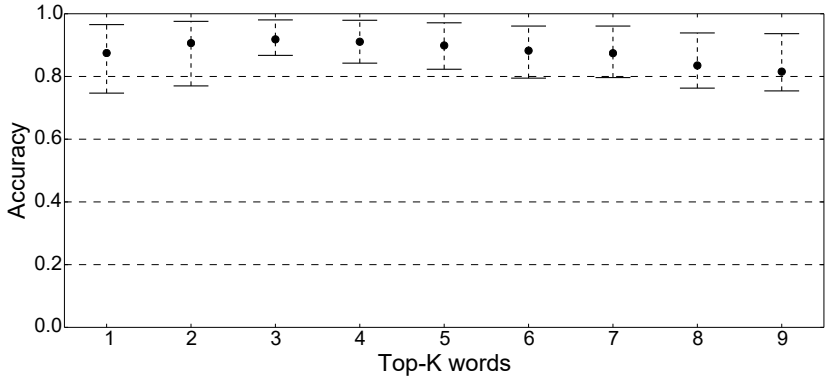


Fig. 4.4 K vs. accuracy. The circles indicate median values and the bars indicate maximum/minimum values, respectively.

the minimum variance. As we increase K, the accuracy is degraded; i.e., as K increases, the less relevant the keywords become. Given these observations, in the following analysis, we adopt $K = 3$ in classifying the document. We note that the chosen threshold works nicely for the external dataset provided by the authors of WHYPER [79]. We will report the results in Section 4.4.3.

Accuracy of Text Classification

We now evaluate the accuracy of our text classifier. To measure the accuracy, we use several metrics. First, TP, TN, FP, and FN represents number of true positives, number of true negatives, number of false positives, and number of false negatives, respectively. We also use three derivative metrics: accuracy (ACC),

Table 4.5 Accuracy of our approach ($K = 3$) for the 6 of labeled datasets.

Resource	Lang	TP	TN	FP	FN	ACC	PPV	NPV
Location	EN	118	591	20	10	0.959	0.855	0.983
	CN	23	826	2	15	0.980	0.920	0.982
Contact	EN	177	396	53	31	0.872	0.770	0.927
	CN	64	535	9	38	0.927	0.877	0.934
Camera	EN	206	284	5	74	0.867	0.976	0.802
	CN	98	575	8	59	0.909	0.925	0.907

Positive predictive values (PPV), and Negative predictive values (NPV), which are defined as

$$ACC = \frac{TP + TN}{TP + TN + FP + FN},$$
$$PPV = \frac{TP}{TP + FP},\ NPV = \frac{TN}{TN + FN},$$

respectively. PPV and NPV measure how many of descriptions classified as positive/negative are actually positive/negative. These measures are suitable to our requirements because we aim to derive the answers of our research question by studying the characteristics of classified descriptions. Therefore, we expect that these measures have high values.

Table 4.5 presents the results of performance evaluation. In both languages, the observed accuracy was good for all categories; e.g., ACCs were 0.87–0.98. Also, in most cases, NPVs were larger than 0.9. Since one of our objectives is to understand the reasons why text descriptions fail to refer to access permissions, the high number of NPVs is helpful, because it indicates that majority of descriptions classified as negative are actually negative. In summary, our scheme was validated to enable automatic classification of text descriptions into the two categories with good accuracy. It works well for both languages, English and Chinese.

Robustness

To validate the robustness of our approach, we use the external labeled dataset [95], which is provided by the authors of the WHYPER framework [79]. Since the dataset also includes the outcomes of the WHYPER framework, we can directly compare

Table 4.6 Statistics of the WHYPER datasets.

	Contact	Calendar	Audio
# of positive samples	107	86	119
# of negative samples	83	110	81

Table 4.7 Comparison of accuracy of ACODE ($K = 3$), WHYPER semantic analysis (WHYPER), and WHYPER keyword (WKW).

Resource	method	TP	TN	FP	FN	ACC	PPV	NPV
Contact	ACODE	96	63	20	11	0.837	0.828	0.851
	WHYPER	92	77	6	15	0.889	0.939	0.837
	WKW	95	46	37	12	0.742	0.720	0.793
Calendar	ACODE	77	98	12	9	0.893	0.865	0.916
	WHYPER	81	99	11	5	0.918	0.880	0.952
	WKW	84	60	50	2	0.735	0.627	0.968
Audio	ACODE	95	57	24	20	0.742	0.720	0.793
	WHYPER	103	69	12	16	0.860	0.896	0.812
	WKW	113	38	43	6	0.755	0.724	0.864

the performance of the two frameworks. Since the dataset consists of a set of labels for each sentence, we reconstructed original descriptions from the sentenses and assign labels to the descriptions; i.e., if a description consists of at least one sentence that declares the use of a permission, the description is labeled as positive, otherwise labeled as negative. Table 4.6 summarizes the dataset*[1]. All the descriptions are written in English.

Table 4.7 shows the comparison of performance of the ACODE framework and the WHYPER framework in classifying descriptions. Our results show that the performance of the ACODE framework is comparable with that of the WHYPER framework. Especially, the delta for NPV, which is the most important metrics for our study, is less than 0.04 for all the three cases. We also notice that the keyword-based approach used in the WHYPER paper (WKW in the table) had high false positives. We conjecture that the high false positives are due to the nature

*[1]We derived these numbers by analyzing the dataset [95]

of extracted keywords, which include some generic terms such as data, event, and capture.

Notice that the WHYPER dataset consists of higher fractions of positive descriptions, compared to ours. This may reflect the fact that the apps used for WHYPER study were collected from the top-500 free apps; i.e., it is likely the top apps were built by skilled developer and had informative descriptions. In contrast, our datasets consist of larger fractions of negative samples. Since our datasets were collected from entire app space, they consist of various apps, including the ones that failed to add informative descriptions due to the reasons that will be described in the next section. Despite this potential difference in the population of datasets, our framework established good accuracy among all the datasets.

In summary, we evaluated the accuracy of the ACODE framework using 5 of 11 permissions we considered*[1]. In the following large-scale analysis, we assume that the ACODE framework establishes good accuracy for the rest of permissions as well. The potential effect of the assumption will be discussed in Section 4.5.5.

4.5 Analysis of Codes and Descriptions

Using the ACODE framework, we aim to answer our research question **RQ** shown in Section 4.1. We first describe the details of the data sets we used for our analysis, in section 4.5.1. Then, we apply our code analysis to the apps and extract apps with *callable* APIs/URIs of permissions ($C2$, see figure 4.1) in section 4.5.2. Using the extracted apps with callable APIs/URIs of permissions, section 4.5.3 aims to quantify the fractions of apps with text descriptions that successfully inform users about the use of privacy-sensitive resources for each resource. In section 4.5.4 we aim to answer the research question **RQ**. We discuss in-depth analysis to understand the reasons of failures for text descriptions classified as $C4$ in informing users about access permissions. Finally, Section 4.5.5 discusses the limitations of our analysis and evaluation.

*[1]To be precise, we verified 5 of 11 permissions for English and 3 of 11 permissions for Chinese.

Table 4.8 Summary of Android apps used for this work.

	English	Chinese	Data collection periods
Official (Google Play, free)	100,000	0	Apr 2012 – Apr 2014
Official (Google Play, paid)	10,000	0	May 2016 – June 2016
Third-party (Anzhi)	0	74,506	Nov 2013 – Apr 2014
Third-party (Nduoa)	0	25,494	Jul 2012 – Apr 2014

Table 4.9 Numbers of extracted apps for each category.

Official market apps (free 100k)

	Personal data				SMS		Hardware resources		System resources		
	Location	Accounts	Contacts	Calendar	SMS (read)	SMS (send)	Camera	Audio	Get tasks	Kill bg processes	Write setting
Permission (C0)	25026	9943	6962	1893	1352	3471	10232	5204	4646	409	2873
API/URI (C1)	23390	6948	6177	333	526	2043	7173	4621	3433	248	1954
Callable (C2)	18165	3933	4238	100	287	1567	6141	3297	1737	208	1744

Official market apps (paid 10k. All values are multiplied by 10 for comparison)

	Personal data				SMS		Hardware resources		System resources		
	Location	Accounts	Contacts	Calendar	SMS (read)	SMS (send)	Camera	Audio	Get tasks	Kill bg processes	Write setting
Permission (C0)	11110	6990	4520	720	710	1350	6220	5090	2950	470	3360
API/URI (C1)	9510	3760	3710	490	190	970	4180	4400	1470	280	2050
Callable (C2)	7100	1790	2760	310	100	860	3120	2250	1020	210	1900

Third-party market apps

	Personal data				SMS		Hardware resources		System resources		
	Location	Accounts	Contacts	Calendar	SMS (read)	SMS (send)	Camera	Audio	Get tasks	Kill bg processes	Write setting
Permission (C0)	40278	6585	9907	394	7686	16204	14581	10745	37436	7457	15249
API/URI (C1)	36885	3148	4863	98	4668	13807	6934	8354	19147	1158	11564
Callable (C2)	32122	1542	3429	66	4185	12355	6139	6147	15447	957	10299

4.5.1 Data sets

We collected Android apps from the official marketplace [96] and two other third-party marketplaces [97, 98]. All these marketplaces have huge user bases. After collecting mobile apps, we first pruned samples that are corrupt or have zero length text descriptions. From the rest of the samples, we randomly picked 100,000 free apps for each type of markets. In addition, we also collected 10,000 paid apps, which enable us to study whether the paid apps have more "informative" text descriptions. Table 4.8 summarizes the data sets we collected. To simplify the interpretation of analyses, we assigned different languages, English and Chinese, to the official and third-party marketplaces. Note that we have already shown that our text description classification scheme works well for both languages.

4.5.2 Extracting apps with callable APIs/URIs of privacy-sensitive resources

Table 4.9 presents the results of our code analysis. Overall, many applications require permission of location. As we will detail later, many of these are apps that use ad libraries. This observation agrees with the fact that paid apps contained fewer location permissions/APIs than free apps. Interestingly, the popularity of personal data resource requirements is almost identical across markets. The most popular is location, second is contact, third is accounts, and fourth is calendar. Generally, third-party markets tend to require/use more permissions than the official market. This may correlate to the existence of defense mechanisms installed on the official marketplace – Bouncer [99].

Another useful finding we can extract from the results is that over privilege ($C0-C1$) is observed commonly across the categories. Also, there are non-negligible numbers of apps that have code to use permissions but cannot be called ($C1 - C2$). This often occurs when a developer incorporates an external library into an app; the library has many functions, including APIs/URIs of permissions, but the app does not actually call the APIs/URIs. Our code analysis can prune these applications

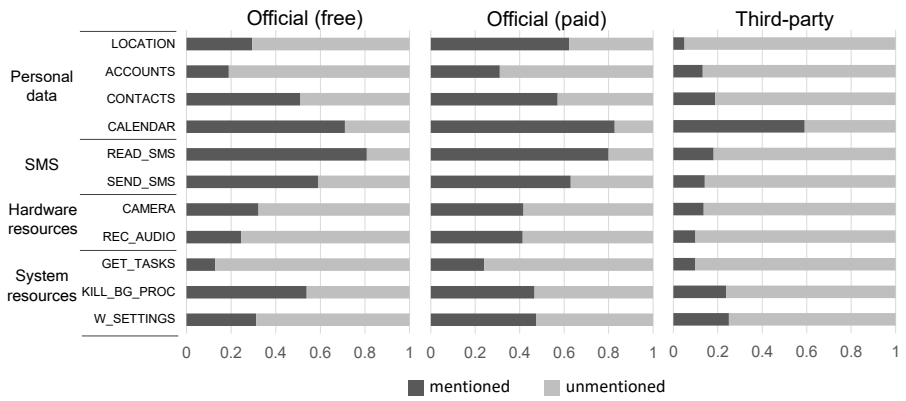


Fig. 4.5 Fractions of descriptions that refer to a permission. Populations are $C2$ apps shown in Table 4.9; e.g., of the 18,165 of official market free apps with callable functions that request location permission, roughly 30% of them mentioned the use of location in the description.

from further analysis.

Overprivilege ratios are especially high for account and contact permissions in the third party marketplaces and for camera, calendar, and kill background processes permissions in both markets. Careful manual inspection revealed that these cases can be attributed to misconfiguration on the part of developers; i.e., the Intent issue discussed in section 4.3.2. Such apps were pruned by the second filter. We also note that these apps do not need to declare permissions because the permissions are misconfigurations. These observations agree with the work performed by Felt et al. [71]. Although our scheme pruned those applications, the pruning did not affect the analysis because the pruned apps are unlikely to exhibit special characteristics in their text descriptions.

4.5.3 Analysis of apps with callable APIs/URIs for a permission.

Using apps that include callable APIs/URIs for a permission ($C2$ in Table 4.9), we analyzed their text descriptions. Figure 4.5 presents the results. We first notice that fractions of positive text descriptions are higher for official market apps. This can

be considered natural, given that official market is more restrictive. We also notice that some resources such as `CALENDAR` for both markets and `SMS` permissions and the `KILL_BG_PROC` (kill background process) permission for the official market are well described in their descriptions.

For the free apps of official market, `GET_TASK` and `ACCOUNTS` were the permissions that were less described (15–20%). In contrast, `READ_SMS` and `CALENDAR` were the permissions that were well described (70–80%). These results are consistent with intuition that permissions that are directly associated with user actions tend to be well described. Overall, our impression is that for the official market, the fractions of proper descriptions are higher than expected. Especially, paid apps have the informative text descriptions about the use of privacy sensitive resources, which agrees with our general expectation that paid products may have higher quality/safety than free products. If the descriptions of remaining apps were improved, the text description could serve as a good source of information to let users know about sensitive resources.

Finally, we note that the descriptions of apps collected from official market was only English, while the descriptions of apps collected from third-party market was only Chinese. Therefore, we cannot tell if the observed differences are due to the market or the language. We leave the issue for future work.

4.5.4 Answers to the Research Question

To answer the research question **RQ**, we performed the manual inspection to the extracted apps that fail to refer to use of permissions. The methodologies of the manual inspection are described below. Given a permission, e.g., `Camera`, we fist identify Java classes that include the APIs associated with the permission. From the identified class, we can extract a package name such as `/com/google/android/foo/SampleCameraClass.java`, which is segmented into a set of words, com, google, android, foo, and SampleClass. By analyzing the package name words for apps that fail to refer to use of the permission, we can find intrinsic words that are associated with specific libraries such as “zxing” used for handling QR code or service names such as “cordova”, which is an app building framework. In addition,

we can analyze developer certificates included in app packages. We also apply dynamic analysis of the apps when we need to check how the permission is used. Using the methodologies, we classified such apps into the four categories. For each category, we extracted reasons why text descriptions fail to refer to permissions.

(1) **App building services/frameworks**

Through the analysis of package names of apps, we noticed that many of apps were developed with cloud-based app building services, which enable a developer to create a multi-platform app without writing code for it. Examples of cloud-based app building services are SeattleClouds, iBuildapp, Appsbar, appbuilder, and biznessapps. Similarly, many of apps were developed with mobile app building frameworks, which also enable a developer to create a multi-platform app easily. Examples of such mobile app building frameworks are Apache Cordova (Phonegap) and Sencha. These services/frameworks provide a simple and intuitive interface to ease the processes of building a mobile app.

Among many such services/frameworks, we found a few services/frameworks that generate apps that *unnecessarily* install many permissions, and put callable APIs/URIs for the permissions into the code. Since a developer using such a service/framework cannot change that setting, it is likely that even the developer is not aware of the fact that app install the permissions with callable APIs/URIs; hence, it is less likely the developer writes about the permissions in the description.

Figure 4.6 shows CDFs of number of permissions per application. First, apps collected from the official market have small number of permissions among the 11 permissions; i.e., more than 80% of apps had zero permissions. They had other generic permission such as Internet. Second, we considered an intrusive cloud-based app building service and one of the popular app building frameworks. Both cases tend to install a large number of permissions. Especially, roughly half of the apps that were built with the intrusive cloud-based app building service had a fixed number of permissions (4 out of 11). We carefully inspected these apps, and found that many permissions such as record audio were unnecessarily installed by the services/frameworks.

We revealed that the apps built by the intrusive cloud-based app building services

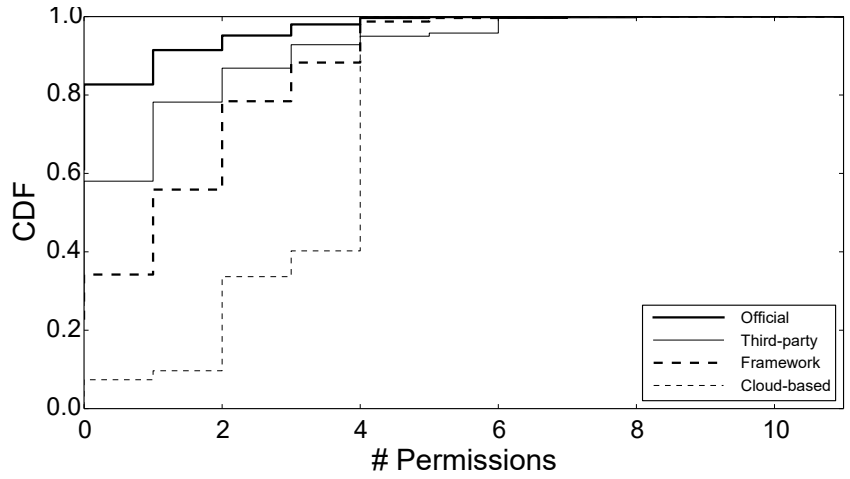


Fig. 4.6 CDFs of number of permissions per application. The 11 permissions listed in Table 4.1 are used.

are popular in official market, but not popular in third-party market. In the official market, more than 65% of apps that failed to refer to use of `record audio` were developed with these services. Similarly, more than 25% of apps that failed to refer to use of `contact list` were developed with these services. We also observed non-negligible number of such apps in other resources; i.e., 5% for location and 10% of camera. For app building frameworks, one of the frameworks accounted for more than 28% of apps that failed to refer to use of `record audio` in the third-party market. In fact the permission was not necessary for the apps.

We also note that unnecessarily installed permissions on a framework such as phonegap, which is HTML5-based mobile app building framework, could bring additional threats because such permission can be abused through various channels of Cross-Site Scripting attacks [100].

(2) **Prolific developers**

Through the analysis of distributions of number of apps per developer certificate, we noticed that a very few number of developers accounted for a large number of descriptions without mention of privacy-sensitive resources. We call such developers "prolific developers". For instance, five prolific developers published 47% of third-party market apps that fail to refer to `send SMS`. We applied eleven popu-

lar commercial anti-virus scanners to the apps with SMS permission, and checked whether either of scanner detected the types of application. If at least one scanner detected an app as malware/adware, we marked it as malware/adware. We found that majority of the apps with unmentioned SMS permission were malware/adware and have been removed from the market later. There are other cases. Three prolific developers published 38% of third-party market apps that fail to use of `kill background processes`. Another three prolific developers published 32% of third-party market apps that fail to use of `write setting`. We carefully inspected these apps, and found that they do not have any reasons to use the permissions. Although not conclusive, we conjecture that these prolific developers likely reuse their own code for building a large number of apps; i.e., they tend to include unnecessary permissions/code.

(3) **Secondary functions**

Through the careful analysis of descriptions that failed to refer to permissions, we found several secondary functions that tend to be unmentioned. For instance, several apps have functions to share information with friends, e.g., scores of games. In many cases, such functions require to access contact list. However, such activity is often unmentioned in the descriptions because it is an optional function. Another example is map-based apps that require to access the `write setting` permission to enable location positioning service such as GPS or Wi-Fi. Such map-based apps accounted for 44% of apps that failed to refer to `write setting`. Among several cases, the most notable one was barcode reader, which requires access to `camera` device. Although there are several barcode reader apps, majority of apps with barcode reader function are shopping apps or social networking apps. Since the barcode reader is not a primary function for those apps, it tends to be unmentioned in their descriptions. To study the impact of such cases, we extracted apps that use barcode libraries such as ZXing [101] or ZBar [102]. We found that in the official market, more than 53% of apps that failed to refer to use of camera had barcode reader libraries in their code. In the third-party market, more than 66% of such apps had barcode libraries. Mobile application distribution platform providers may want to support exposing the use of privacy-sensitive resources by functions that tend to

be unmentioned.

(4) **Third-party libraries**

Just as the source of many vulnerabilities are in third-party libraries [103], unintentional use of privacy-sensitive resources can also be caused by the libraries. For instance, it is well known that ad libraries make use of resources of location or account information for establishing targeted advertisement [73]. Another example of third-party libraries are log analysis libraries and crash analysis libraries. These libraries make use of `get task` permission and location information. We analyzed apps that have callable location APIs/URIs and text descriptions that do not refer to the location permission. We found that in the official market, more than 62% of such apps use ad libraries. In the third-party market, more than 80% of such apps used ad libraries. Similarly, in the third-party market, more than 20% of apps that failed to refer to location permission used access analysis libraries. Thus, if a developer uses these third-party libraries, it is likely that the description of the app fails to refer to the permission unless the developer explicitly expresses it.

4.5.5 Threats to Validity

This section discusses several limitations of our analysis and evaluation.

Static code analysis

Although we developed an algorithm to check whether privacy-sensitive APIs/URIs are callable, we are aware of some limitations. First, although the algorithm can detect the callability of APIs/URIs, we cannot precisely ensure that they are actually called. Second, our static code analysis cannot dynamically track assigned program code at run-time, such as reflection. Third, as Poeplau et al. [104] revealed, some malware families have the ability to self-update; i.e., after installation, an app can download the new version of itself and load the new version via `DexClassLoad`. Employing dynamic code analysis could be a promising solution to these problems. However, other challenges may include scalability and the creation of test patterns for UI navigations [105, 106]. As we mentioned earlier, we adopted static analysis because our empirical study required analysis of a huge volume of applications.

On the other hand, we note that static code analysis has a chance to extract *hidden* functions that cannot be explored by a dynamic analysis. We leave these challenges for our future work.

Accuracy of the keyword-based approach

As we mentioned earlier, we evaluated the accuracy of the ACODE framework using 5 of 11 permissions we considered. Our assumption is that the ACODE framework establishes good accuracy for the rest of 6 permissions. However, there may be a concern that the keyword-based approach works better for some permissions more than others. We note that some of the results derived in Section 4.5.4 were based on permissions for which we did not evaluate the accuracy; e.g., `SEND_SMS`, `KILL_BG_PROC`, and `GET_TASKS`. Therefore, the results might have threats to validity. A simple solution to address the concern is to extend the labeled dataset, however, we were not able to perform the additional experiments due to the high cost of labeling descriptions written in two languages. Although not conclusive, we note that we have validated that the descriptions were correctly classified through the manual inspection, using randomly sampled apps; i.e., the obtained results were partially validated.

4.6 Discussion

In this section, we discuss the feasibility and versatility of the ACODE framework. We also outline several future research directions that are extensions of our work.

4.6.1 User experience

In this study, we asked participants to read whole sentences carefully, regardless of the size of the text description. In a real-user setting, users might stop reading a text description if it is very long. Studying how the length of text descriptions or the placement of permission-related sentences affect user awareness is a topic for future work. In addition to text descriptions, mobile software distribution platforms provide other information channels, such as meta data or screenshots of an app. As

users may also pay attention to these sources of information, studying how these sources provide information about permissions is another research challenge we are planning to address.

4.6.2 Cost of analysis

Because this work aims to tackle with a huge volume of applications, we adopt light-weight approaches; static code analysis (instead of dynamic code analysis) and keyword-based text analysis (instead of semantic analysis). In the followings, we detail the cost of our approach. The cost of data analysis with the ACODE framework can be divided into two parts: the static code analyzer and the text descriptions analyzer. For the static code analyzer, the most expensive task is the function call analysis because we first need to build function call trees to study whether an API is callable. Our empirical study showed that the task of function call analysis for an application took 6.05 seconds on average. We note that the tasks can be easily parallelized. By parallelizing the tasks with 24 of processes on a commodity PC, we were able to process 200 K apps within a single day. For the text description analyzer, collecting label was the most expensive task. On average, a single participant labeled 1,500 of descriptions within 10 hours. However, once we get the performance evaluation of our approach, we do not need to employ the task again because our work does not need manually-labeled samples. Since we adopt keyword-based approach, analyzing hundred thousands of descriptions was quite fast.

Overall, all the tasks can be completed within a single day, and we can further accelerate the speed if this is desired. As our objective is not to perform the analysis in real-time, we believe that the cost of performing analyses with the ACODE framework is affordable.

4.6.3 Permissions that should or should not be mentioned.

Android OS manages several permissions with a protection level defined as “dangerous,” which means “a higher-risk permission that would give a requesting application

access to private user data or control over the device that can negatively impact the user [107]." Ideally, users should be aware of all these dangerous permissions. The dangerous permissions can be broadly classified into two categories: for users and for developers. Permissions for users include read/write contacts, access fine location, read/write calendar, read/write user dictionary, camera, microphone, Bluetooth, and send/read SMS. The resources analyzed in this chapter are the permissions aimed at users. Permissions for developers include set debug app, set process limit, signal persistent processes, reorder tasks, write setting, and persistent activity.

Permissions for users are intuitively understandable. Thus, they should be described in the text descriptions. Permissions for developers are difficult for general users to understand; thus, describing them may be confusing. As describing these permissions could even distract users' attention from the text descriptions, they should *not* be mentioned in the text descriptions. For such dangerous permissions aimed at developers, we need to develop another information channel that lets users know about the potential threats in an intuitive way. We note that the ACODE framework can be used to identify dangerous permissions that are least mentioned. Knowledge of such permissions will be useful to develop a new information channel.

4.7 Related work

Researchers have studied mobile apps from various viewpoints, including issues of privacy, permission, and user behavior. In this section, we review the previous studies along four axes: system-level protection schemes, large-scale data analyses, user confidence and user behavior, and text descriptions of mobile apps.

4.7.1 System-level protection schemes

As a means of protecting users from malicious software, several studies have proposed install-time or runtime protection extensions that aim to achieve access control and application isolation mechanisms such as [108–112]. Kirin [108] performs lightweight certification of applications to mitigate malware at install-time based on a conservative security policy. With regard to install-time permission policies

and runtime inter-application communication policies, SAINT [109] provides operational policies to expose the impact of security policies on application functionality, and to manage dependencies between application interfaces. TaintDroid [110] modifies the operating system and conducts dynamic data tainting at runtime in order to track the flow of sensitive data to detect when this data is exfiltrated. Quire [113] is defense mechanisms against privilege escalation attacks with inter-component communication (ICC). Finally, SEAndroid [112] brings flexible mandatory access control (MAC) to Android by enabling the effective use of Security Enhanced Linux (SELinux).

While the above studies improved the system-level security and privacy of smartphone, this work attempts to address the problem from a different perspective – understanding the effectiveness of text description as a potential source of information channel for improving users' awareness of privacy.

4.7.2 Large-scale data analyses

Several researchers have conducted measurement studies to understand how many mobile apps access to private resources and how they use permissions to do so [70–73]. A survey report published by Bit9 [70] included a large-scale analysis of Android apps using more than 410,000 of Android apps collected from the official Google Play marketplace. Through the analysis, they revealed that roughly 26% of apps access personal information such as contacts and e-mail, 42% of apps access GPS, and 31% of apps access phone calls or phone numbers. Book et al. [73] analyzed how the behavior of the Android ad library and permissions have changed over time. Through the analysis of 114,000 apps collected from Google Play, they found that the use of most permissions has increased over time, and concluded that permissions required by ad libraries could expose a significant weakness in user privacy and security. From the perspective of dynamic code loading, Poeplau et al. [104] conducted an analysis of 1,632 popular apps, each with more than 1 million installations, and revealed that 9.25% of them are vulnerable to code injection attacks.

4.7.3 User confidence and user behavior

Several works on user confidence and user behavior discuss users' installation decisions [77, 114–116]. Refs. [115, 116] studied user behavior in security warnings, and revealed that most users continue through security warnings. Good et al. [114] conducted an ecological study of computer users installing software, and found that providing vague information in EULAs and providing short notices can create an unwarranted impression of increased security. Chin et al. [77] studied security and privacy implications of smartphone user's behaviors based on a set of installation factors, e.g., price, reviews, developer, and privacy. Their study implicates user agreements and privacy policies as the lowest-ranked factors for the privacy. As these studies on user confidence and behavior suggest, user agreements or privacy policies are not effectively informing consumers about privacy issues with apps. Centralized mobile software distribution platforms should provide mechanisms that improve privacy awareness so users can use apps safely and confidently. We believe that our findings obtained using the ACODE framework can be used to complement these studies.

4.7.4 Text descriptions

As mentioned in section 4.1, only a few works have focused on text descriptions of mobile apps [78–81]. The WHYPER framework [79] is the pioneering work that attempted to bridge the semantic gap between application behaviors and user expectations. They applied modern NLP techniques for semantic analysis of text descriptions, and demonstrated that WHYPER can accurately detect text sentences that refer to a permission. Qu et al. [81] indicated an inherent limitation of the WHYPER framework, i.e., the derived semantic information is limited by the use of a fixed vocabulary derived from Android API documents and synonyms of keywords there. To overcome the issue, they proposed the AutoCog framework based on modern NLP techniques extracting semantics from descriptions without using API documents. The key idea behind their approach is to select noun-phrase based

governor-dependent pairs related to each permission. They demonstrated that the AutoCog framework moderately improved performance as compared to the WHYPER framework. Gorla et al. [80] proposed the CHABADA framework, which can identify anomalies automatically by applying an unsupervised clustering algorithm to text descriptions and identifying API usage within each cluster. Like our work, CHABADA uses API functions to identify outliers. On the other hand, the aim of ACODE is *not* to find anomalies, but to quantify the effectiveness of text descriptions as a means of making users aware of privacy threats. To this end, using a simple keyword-based approach, the ACODE framework attempts to assess the reasons why text descriptions do not refer to permissions. As we revealed, the performance of our approach is comparable with that of the WHYPER framework. We also note that the ACODE framework is more fine-grained than CHABADA since ACODE checks whether API functions/URIs found in code are callable by employing function call analysis. Finally, Lin et al. [78] studied users' expectations related to sensitive resources and mobile apps by using crowdsourcing. They asked participants to read the provided screenshots and text description of an app, and asked several questions to investigate users' perceptions of the app as related to privacy-sensitive resources. They concluded that users' expectations and the purpose for using sensitive resources have a major impact on users' subjective feelings and their trust decisions. This observation supports the importance of improving users' privacy awareness on mobile software distribution platforms.

We summarize the differences among the above three studies, and our own in table 4.10. In addition to the technical differences, our work is distinguishable from other studies in its large-scale empirical analysis, which spans across 11 of distinct permissions, two market places, both free and paid, and more than 200K of text descriptions written in two different natural languages.

Recently a few works [117, 118] take an alternative approach to use a text description as an information channel of privacy threats. They aim to generate security-centric text descriptions automatically by using the static analysis of app code. This approach is useful to describe the potentially malicious behavior of apps. It can also extract the secondary functions, which are not described in the text description

Table 4.10 Comparison between related works.

	ACODE	WHYPER	AutoCog	CHABADA	Lin et al. [10]
objective	Understanding inconsistency between codes and descriptions	Identifying sentences that refer to a permission	Assessing description-to-permission fidelity of applications	Identifying outlier apps	Understanding user expectation on sensitive resources
# of apps	free 200,000 paid 10,000	581	83,656	32,308	134
# of studied permissions	11	3	11	N/A	4
markets	Official, Third-party	Official	Official	Official	Official
languages	English, Chinese	English	English	English	English
code analysis	Function call tree analysis	Permission check	Permission check	API analysis	Permission check
description analysis	Keyword-based	Semantic analysis	Semantic analysis	Topic model	N/A

written by a developer. The drawback of the approach is that they cannot cope with the cases of app building services/frameworks and some third-party libraries that often inject unnecessary APIs which are never called from the app; i.e, dead code. For such cases, the automatically generated descriptions will not agree with the actual behavior. Addressing the issue needs advanced program analysis such as dynamic analysis.

4.8 Conclusion

By applying the ACODE framework to 200,000 free apps and 10,000 paid apps collected from both official and third-party marketplaces, our analysis across the 11 distinct resources revealed four primary factors that are associated with the inconsistencies between text descriptions and use of privacy-sensitive resources: (1) existence of app building services/frameworks that tend to add API permissions/code unnecessarily, (2) existence of prolific developers who publish many applications that unnecessarily install permissions and code, (3) existence of secondary functions that tend to be unmentioned, and (4) existence of third-party libraries that access to the privacy-sensitive resources. We also found that paid apps generally have more informative text descriptions than free apps, which probably reflects the developers'

motivation to achieve a high number of downloads.

We believe that our work provides an important first step toward improving users' privacy awareness on mobile software distribution platforms. For instance, developers of app building services/frameworks can use our findings to check the behaviour and deployment of their products. Individual mobile app developers can pay attention to our findings when they write text descriptions or use third-party libraries. And mobile software distribution platform providers can pay attentions to all the potential reasons that lead to the inconsistencies between user expectations and developer intentions. Based on the findings revealed by the ACODE framework, they may be able to come up with new information channels that effectively inform users about the use of privacy-sensitive resources.

Chapter 5

Conclusion

5.1 Summary of Thesis

The explosive diffusion of Internet devices brings both enormous convenience and novel risks to users. Modern attackers exploit information outside computer systems beyond the layers, making traditional defense techniques difficult to apply. We aimed to identify novel privacy threats for Internet users and measured their impact through conceptual and empirical approaches.

The web, mobile, and IoT we have covered in this thesis are major ways for consumers to access the Internet. To identify novel privacy threats beyond the layers, this thesis presented attack and defense scenarios that extract information from outside computer systems with physical, network, and human layers. We discussed new methods of leaking real location information and identity of Internet users and how to detect applications that access privacy-sensitive data against user expectations and elucidate the cause. The web, mobile, and IoT we covered in this thesis are major ways for consumers to access the Internet.

In Chapter 2, we introduced a novel attack framework called RouteDetector, which leverages spatio-temporal regularity of human mobility by targeting train trips of users. Our results quantitatively support that such as attack is feasible. This contribution indicated that the hardware resources that leak privacy are not only the typical ones such as GPS. The overlooked resource of physical sensors leaks

human-location information.

In Chapter 3, we presented a practical side-channel attack to identify the social account of a website visitor. By the precise designed of this attack, the 1-bit rough application state estimated from the RTT restored the privacy data of the specified username. Through our efforts, major services and browsers adopted new security features. This contribution goes beyond just preventing such an attack. We have identified a vulnerability in a universal feature in which one user controls the visibility of another through observations from another layer.

In Chapter 4, we presented an analytical method for app markets and recommendations to help non-malicious developers and users communicate. Our analysis revealed the percentage of apps that access privacy-sensitive data against user expectations and four primary factors that are associated with the inconsistencies between text descriptions and use of privacy-sensitive resources. Regardless of how robust a system is, user privacy is threatened when human perception is exploited. Our research provides an important first step toward improving users' privacy awareness on mobile software distribution platforms.

The arms race in cyber security has intensified, expanding the war from cyberspace to another space. Protecting only computer systems is no longer sufficient. Attacks we identified and countermeasures we deployed helped protect users' privacy from a wide range of attacks beyond the layers. We shed light on the privacy-leaking channel that has been overlooked by presenting the concept of new attacks beyond the layers and revealed the actual impact of attacks by conducting large-scale empirical analysis in the wild.

The main contribution of this thesis is not only fixing each vulnerability of services and applications. The countermeasures and guidelines we proposed enable service providers to comprehensively defend against attacks that have a common attack principle. However, there are still potential channels that attackers can exploit. In the next section, we discuss the future directions of our research to reduce privacy threats for Internet users.

5.2 Future Directions

Finally, we present the following three directions for future research.

Expanding Attack Channels to Investigate. We recognize that there are other potential privacy threats beyond the layers. Attackers may be able to estimate privacy data by measuring other physical phenomena such as measuring power consumption, acoustic behavior, and electromagnetic waves. Privacy-protection mechanisms become more comprehensive by expanding the attack channels to investigate. The field of cryptography leading with side-channel attacks will provide insights into our research.

Measuring the Effectiveness of Offensive Research. We published a procedure to reproduce privacy attacks we found as an academic paper along with appropriate countermeasures and guidelines. We believe this activity is beneficial to inform service providers and users of risks and defenses, and is a common approach in the security research community. However, we have not measured how many attacks were actually prevented. We leave the measurement to understand both the positive and negative effects of disclosure for future work.

Considering Useful Aspects of Attack Concepts. The attacks we have discussed in Chapters 2 and 3 are practical for user profiling. Location data and attributes of the user linked to the social account are useful for efficient marketing and improving service quality. With user consent, our attacks can be useful for legitimate service providers rather than as actual attacks. Considering the practical aspects of privacy attacks is an issue for future.

Acknowledgement

Foremost, I express my sincere thanks to my supervisor, Prof. Tatsuya Mori, for the continuous support of my Ph.D. thesis and research. I joined as one of the first students in his laboratory when I was an undergraduate. Since then, I learned almost everything as a researcher, including the basics of cybersecurity and network systems, how to read and write academic papers, how to design experiments, how to make a presentation, and how to compile LaTeX. Above all, the best harvest of my student life is *grit*, which is passion and perseverance for long-term goals. Although I have submitted papers to top-tier conferences many times and have been rejected each time over the years, we got accepted for the NDSS in the final year of my Ph.D. course. Going forward, when I encounter difficulties, I will solve them with the grit Prof. Mori instilled in me.

Next, I thank my sub-advisor, Prof. Hironori Washizaki, for his grateful support. He gave me insightful advice from a software engineering professional's point of view. I also thank the referees for my doctoral thesis. Prof. Masato Uchida gave me meaningful comments and reinforced my research theoretically and increased the generality of the research. Prof. Katsunari Yoshioka is a premier cybersecurity researcher, and I am constantly receiving practical comments from him to develop my research. In addition, I would like to thank Prof. Tetsuya Sakai. He gave me the necessary knowledge to properly implement natural language processing for my first study to analyze the Android app's descriptions. This work motivated me to become a researcher.

I also thank my colleagues at NTT Secure Platform Laboratories for supporting my research. Especially, Dr. Mitsuaki Akiyama was my mentor during my internship at NTT, and he is currently a research leader. He is always examining how our research

can be applied to the real world from a high perspective. He gives me insights to maximize my research output. Mr. Eitaro Shioji is my mentor after I joined NTT. His comments, based on critical and logical thinking, have improved the quality of my research. The reason I got into the research field of offensive security with confidence was that he affirmed and developed my ideas. Dr. Takeshi Yagi, who currently belongs to NTT Security Japan, has given me a lot of great advice not only on my research but also on my attitude as a researcher. Mr. Fumihiro Kanei, Dr. Yuta Takata, and Mr. Toshiki Shibahara helped my Android app research greatly. The constructive comments given by Dr. Daiki Chiba and Ms. Asuka Nakajima have improved the quality of my papers and conference proposal. Other team members also contributed to my work through daily discussions. I am very happy to have had them as my colleagues. In addition, I extend my thanks to thank my previous and current supervisors in NTT, Mr. Takeo Hariu, Mr. Takeshi Yada, Mr. Kazuhiro Hayakawa, and Mr. Kunio Hato for encouraging my research activities.

The members of the Network Security Laboratory (NSL) are irreplaceable to me. Mr. Yuta Ishii, Dr. Bo Sun, and Mr. Keito Sasaoka are co-authors of my research paper and greatly supported my experiments. Without the discussions with my research advisor Dr. Mitsuhiro Hatada, I could not have written this doctoral thesis. The practical techniques that I learned from the CTF team, m1z0r3, helped me to identify novel threats in the wild. Daily communication with the NSL members was the greatest mental support for me.

Lastly, I would like to thank my parents for their special support at all times. Regardless of the number of times that I seemed likely to go astray, they were always patient and helped me. The world of computers and the Internet—which my mother introduced to me during my childhood, though she then became concerned that it would hinder my studies—is an area that I plan to study throughout my life.

Bibliography

[1] "Strategy analytics: Internet of things now numbers 22 billion devices but where is the revenue?." https://news.strategyanalytics.com/press-release/iot-ecosystem/strategy-analytics-internet-things-now-numbers-22-billion-devices-where.

[2] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," IEEE Security & Privacy, vol.5, no.2, pp.32–39, 2007.

[3] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: malware analysis via hardware virtualization extensions," Proceedings of the 15th ACM conference on Computer and communications security, pp.51–62, ACM, 2008.

[4] L.K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the {OS} and dalvik semantic views for dynamic android malware analysis," Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12), pp.569–584, 2012.

[5] Y.M.P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: A novel honeypot for revealing current iot threats," Journal of Information Processing, vol.24, no.3, pp.522–533, 2016.

[6] M. Akiyama, T. Yagi, T. Yada, T. Mori, and Y. Kadobayashi, "Analyzing the ecosystem of malicious url redirection through longitudinal observation from honeypots," Computers & Security, vol.69, pp.155–173, 2017.

[7] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J.A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," 26th {USENIX} Security Symposium ({USENIX} Security 17), pp.1093–1110, 2017.

[8] C. Moore, "Detecting ransomware with honeypot techniques," 2016 Cybersecurity and Cyberforensics Conference (CCC), pp.77–81, IEEE, 2016.

[9] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," A Practical Guide, 1st Ed., Cham: Springer International Publishing, 2017.

[10] "California consumer privacy act." https://www.caprivacy.org/.

[11] "Facebook security breach exposes accounts of 50 million users - the new york times." https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html.

[12] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, *et al.*, "Meltdown: Reading kernel memory from user space," 27th {USENIX} Security Symposium ({USENIX} Security 18), pp.973–990, 2018.

[13] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, *et al.*, "Spectre attacks: Exploiting speculative execution," 2019 IEEE Symposium on Security and Privacy (SP), pp.1–19, IEEE, 2019.

[14] "Spam and phishing in q2 2019 | securelist." https://securelist.com/spam-and-phishing-in-q2-2019/92379/.

[15] L. Cai and H. Chen, "TouchLogger: Inferring Keystrokes On Touch Screen From Smartphone Motion," The 6th USENIX Workshop on Hot Topics in Security (HotSec), 2011.

[16] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "ACCessory: Password Inference using Accelerometers on Smartphones," The Twelfth Workshop on Mobile Computing Systems and Applications (HotMobile), 2012.

[17] Z. Xu, K. Bai, and S. Zhu, "TapLogger: Inferring User Inputs on Smartphone Touchscreens Using On-board Motion Sensors," The fifth ACM conference on Security and Privacy in Wireless and Mobile Networks, 2012.

[18] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R.R. Choudhury, "TapPrints: Your Finger Taps Have Fingerprints," The 10th International Conference on Mobile Systems, Applications, and Services (MobiSys), 2012.

[19] S. Dey, N. Roy, W. Xu, R.R. Choudhury, and S. Nelakuditi, "AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable," The 2014 Network and Distributed System Security (NDSS) Symposium, 2014.

[20] Y. Michalevsky, D. Boneh, and G. Nakibly, "Gyrophone: Recognizing Speech from Gyroscope Signals," The 23rd USENIX Security Symposium, 2014.

[21] A. Das, N. Borisov, and M. Caesar, "Do You Hear What I Hear?: Fingerprinting Smart Devices Through Embedded Acoustic Components," The 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2014.

[22] Y. Michalevsky, G. Nakibly, A. Schulman, and D. Boneh, "Powerspy: Location tracking using mobile device power analysis," CoRR, vol.abs/1502.03182, 2015.

[23] M.C. Gonzalez, C.A. Hidalgo, and A.L. Barabasi, "Understanding individual human mobility patterns," Nature, vol.453, no.7196, pp.779–782, June 2008.

[24] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard, "Sok: Systematic classification of side-channel attacks on mobile devices," arXiv preprint arXiv:1611.03748, 2016.

[25] Y. Gu, A. Lo, and I. Niemegeers, "A Survey of Indoor Positioning Systems for Wireless Personal Networks," IEEE Communications Surveys & Tutorials, 2009.

[26] J. Hua, Z. Shen, and S. Zhong, "We can track you if you take the metro: Tracking metro riders using accelerometers on smartphones," CoRR, vol.abs/1505.05958, 2015.

[27] S. Narain, T.D. Vo-Huu, K. Block, and G. Noubir, "Inferring user routes and locations using zero-permission mobile sensors," 2016.

[28] G. Tsoukaneri, G. Theodorakopoulos, H. Leather, and M.K. Marina, "On the inference of user paths from anonymized mobility data," 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp.199–213, IEEE, 2016.

[29] E. Miluzzo, N.D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S.B. Eisenman, X. Zheng, and A.T. Campbell, "Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe

Application," The 6th ACM conference on Embedded network sensor systems (SenSys), 2008.

[30] S. Bugiel, S. Heuser, and A.R. Sadeghi, "Flexible and Fine-Grained Mandatory Access Control on Android for Diverse Security and Privacy Policies," The 22nd USENIX Security Symposium, 2013.

[31] S. Chakraborty, C. Shen, K.R. Raghavan, Y. Shoukry, M. Millar, and M. Srivastava, "ipShield: A Framework For Enforcing Context-Aware Privacy," The 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2014.

[32] J. Mander, "Internet Users Have Average of 7 Social Accounts." https://blog.globalwebindex.com/chart-of-the-day/internet-users-have-average-of-7-social-accounts/, 2016.

[33] T. Minkus and K.W. Ross, "I Know What You're Buying: Privacy Breaches on eBay," Proc. of PETS, 2014.

[34] G. Mulvenna, "Blocked By Me." https://web.archive.org/web/20170820142006/blockedby.me/.

[35] A. Bortz, D. Boneh, and P. Nandy, "Exposing Private Information by Timing Web Applications," Proc. of WWW, 2007.

[36] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang, "Knowing Your Enemy: Understanding and Detecting Malicious Web Advertising," Proc. of ACM CCS, 2012.

[37] K. Young, "GlobalWebIndex Blog." https://blog.globalwebindex.net/chart-of-the-day/8-in-10-social-networking-on-mobile/, 10 2016.

[38] T.V. Goethem, W. Joosen, and N. Nikiforakis, "The Clock is Still Ticking: Timing Attacks in the Modern Web," Proc. of ACM CCS, 2015.

[39] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A Practical Attack to De-anonymize Social Network Users," Proc. of IEEE S&P, 2010.

[40] V. Jacobson, "Pathchar." https://www.caida.org/tools/utilities/others/pathchar/, 1997.

[41] "Fan Page List." https://fanpagelist.com.

[42] R. Dey, C. Tang, K. Ross, and N. Saxena, "Estimating Age Privacy Leakage in Online Social Networks," Proc. of INFOCOM, 2012.

[43] B. Krishnamurthy, P. Gill, and M. Arlitt, "A Few Chirps about Twitter," Proc. of Workshop on Online Social Networks, 2008.

[44] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, "The anatomy of the facebook social graph," arXiv preprint arXiv:1111.4503, 2011.

[45] G. Kontaxis, M. Polychronakis, A.D. Keromytis, and E.P. Markatos, "Privacy-Preserving Social Plugins," Proc. of USENIX Security, 2012.

[46] S. Lee, H. Kim, and J. Kim, "Identifying Cross-origin Resource Status Using Application Cache," Proc. of NDSS, 2015.

[47] C.Q. Alfred, Q. Zhiyun, and M.Z. Morley, "Peeking into your app without actually seeing it: Ui state inference and novel android attacks.," Proc. of USENIX Security, 2014.

[48] OWASP, "Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet." https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet.

[49] G. Maone, "NoScript Security Suite." https://addons.mozilla.org/en-US/firefox/addon/noscript/, 2016.

[50] Y. Cao, Z. Chen, S. Li, and S. Wu, "Deterministic Browser," Proc. of ACM CCS, 2017.

[51] K. Kufluk and G. Baker, "Protecting user identity against Silhouette." https://blog.twitter.com/engineering/en_us/topics/insights/2018/twitter_silhouette.html.

[52] P.C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," Proc. of CRYPTO, 1996.

[53] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," Proc. of CRYPTO, 1999.

[54] J. Ruderman, "The same origin policy." http://www.mozilla.org/projects/security/components/same-origin.html, 2001.

[55] CERT, "Malicious HTML Tags Embedded in Client Web Requests." https://www.cert.org/historical/advisories/CA-2000-02.cfm?, 2

2000.

[56] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks," Proc. of ACM Symposium on Applied Computing, 2006.

[57] P. Vogt, E. Kirda, C. Kruegel, and G. Vigna, "Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis," Proc. of NDSS, 2007.

[58] B. Stock, S. Lekies, T. Mueller, P. Spiegel, and M. Johns, "Precise Client-side Protection against DOM-based Cross-Site Scripting," Proc of USENIX Security, 2014.

[59] B. Liang, W. You, L. Liu, W. Shi, and M. Heiderich, "Scriptless Timing Attacks on Web Browser Privacy," Proc. of IEEE/IFIP DSN, 2014.

[60] N. Gelernter and A. Herzberg, "Cross-Site Search Attacks," Proc. of ACM CCS, 2015.

[61] Y. Jia, X. Dong, Z. Liang, and P. Saxena, "I Know Where You've Been: Geo-Inference Attacks via the Browser Cache," IEEE Internet Computing, vol.19, pp.44–53, 1 2015.

[62] C. Jackson, A. Bortz, D. Boneh, and J.C. Mitchell, "Protecting Browser State From Web Privacy Attacks," Proc. of WWW, 2006.

[63] J. Ruderman, "css on a:visited can load an image and/or reveal if visitor been to a site." `https://bugzilla.mozilla.org/show_bug.cgi?id=57351`, 2000.

[64] D. Baron, "Preventing attacks on a user's history through CSS :visited selectors." `https://dbaron.org/mozilla/visited-privacy`, 2010.

[65] W3C, "5.11.2 The link pseudo-classes: :link and :visited." `https://www.w3.org/TR/CSS2/selector.html\#link-pseudo-classes`, 2011.

[66] P. Eckersley, "How Unique Is Your Web Browser?." `https://panopticlick.eff.org/static/browser-uniqueness.pdf`, 2009.

[67] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and Defending Against Third-Party Tracking on the Web," Proc. of USENIX NSDI, 2012.

[68] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The Web Never Forgets: Persistent Tracking Mechanisms in the Wild," Proc.

of ACM CCS, 2014.

[69] "Annual number of mobile app downloads worldwide 2021 | Statistic." https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/.

[70] B. Report, "Pausing Google Play: More Than 100,000 Android Apps May Pose Security Risks." https://www.bit9.com/research/pausing-google-play/.

[71] A.P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS '11, pp.627–638, 2011.

[72] A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: user attention, comprehension, and behavior," Symposium on Usable Privacy and Security (SOUPS), 2012.

[73] T. Book, A. Pridgen, and D.S. Wallach, "Longitudinal analysis of android ad library permissions," IEEE Mobile Security Technologies (MoST), 2013.

[74] Y. Zhou and X. Jiang, "Detecting passive content leaks and pollution in android applications," 20th Annual Network & Distributed System Security Symposium (NDSS), Feb. 2013.

[75] J. Kim, Y. Yoon, K. Yi, and J. Shin, "ScanDal: Static analyzer for detecting privacy leaks in android applications," MoST 2012: Mobile Security Technologies 2012, May 2012.

[76] Future of Privacy Forum, "FPF Mobile Apps Study." http://www.futureofprivacy.org/wp-content/uploads/Mobile-Apps-Study-June-2012.pdf.

[77] E. Chin, A.P. Felt, V. Sekar, and D. Wagner, "Measuring user confidence in smartphone security and privacy," Symposium on Usable Privacy and Security (SOUPS), 2012.

[78] J. Lin, S. Amini, J.I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang, "Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing," Proceedings of the 2012 ACM Conference on Ubiquitous Computing, pp.501–510, 2012.

[79] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications," Proceedings of the 22Nd USENIX Conference on Security, pp.527–542, Aug 2013.

[80] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," ICSE'14: Proceedings of the 36th International Conference on Software Engineering, 2014.

[81] V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "Autocog: Measuring the description-to-permission fidelity in android applications," Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, 2014.

[82] M.P. Lewis, ed., Ethnologue: Languages of the World, seventeenth ed., SIL International, Dallas, TX, USA, 2013.

[83] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets," 19th Annual Network & Distributed System Security Symposium (NDSS), Feb. 2012.

[84] "Android asset packaging tool." http://www.kandroid.org/guide/developing/tools/aapt.html.

[85] "PScout: Analyzing the Android Permission Specification." http://pscout.csl.toronto.edu/.

[86] K.W.Y. Au, Y.F. Zhou, Z. Huang, and D. Lie, "Pscout: Analyzing the android permission specification," Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12, pp.217–228, 2012.

[87] "android-apktool." http://code.google.com/p/android-apktool/.

[88] "smali – An assembler/disassembler for Android's dex format." https://code.google.com/p/smali/.

[89] "androguard." https://code.google.com/p/androguard/.

[90] "Kyoto Text Analysis Toolkit." http://www.phontron.com/kytea/.

[91] "Natural Language Toolkit." http://www.nltk.org.

[92] "imdict-chinese-analyzer." https://code.google.com/p/imdict-chinese-analyzer/.

[93] M. Makrehchi and M.S. Kamel, "Automatic extraction of domain-specific stopwords from labeled documents," Proceedings of the IR Research, 30th European Conference on Advances in Information Retrieval, ECIR'08, pp.222–233, 2008.

[94] S.E. Robertson and K.S. Jones, "Simple, Proven Approaches to Text Retrieval," Tech. Rep. 356, University of Cambridge Computer Laboratory, 1997.

[95] "Whyper: Towards automating risk assessment of mobile applications." https://sites.google.com/site/whypermission/.

[96] "Google play." http://play.google.com/.

[97] "Anzhi.com." http://anzhi.com.

[98] "Nduoa market." http://www.nduoa.com/.

[99] J. Oberheide and C. Miller, "Dissecting the android bouncer." SummerCon, Brooklyn, NY., 2012. http://jon.oberheide.org/files/summercon12-bouncer.pdf.

[100] X. Jin, X. Hu, K. Ying, W. Du, H. Yin, and G.N. Peri, "Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation," Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, pp.66–77, 2014.

[101] "Official ZXing ("Zebra Crossing") project home." https://github.com/zxing/zxing.

[102] "ZBar bar code reader." http://zbar.sourceforge.net/.

[103] T. Watanabe, M. Akiyama, F. Kanei, E. Shioji, Y. Takata, B. Sun, Y. Ishi, T. Shibahara, T. Yagi, and T. Mori, "Understanding the origins of mobile app vulnerabilities: a large-scale measurement study of free and paid apps," Proceedings of the 14th International Conference on Mining Software Repositories, pp.14–24, IEEE Press, 2017.

[104] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna, "Execute this! analyzing unsafe and malicious dynamic code loading in android applications," Proceedings of the Network and Distributed System Security Symposium (NDSS), 2014.

[105] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, "Smartdroid: An automatic system for revealing ui-based trigger conditions in android applications," Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '12, pp.93–104, 2012.

[106] A. Gianazza, F. Maggi, A. Fattori, L. Cavallaro, and S. Zanero, "Puppetdroid: A user-centric ui exerciser for automatic dynamic analysis of similar android applications," CoRR, vol.abs/1402.4826, 2014.

[107] "Android developers guide: App manifest – permission." http://developer.android.com/guide/topics/manifest/permission-element.html.

[108] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS '09, pp.235–245, ACM, 2009.

[109] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically rich application-centric security in android," Proceedings of the 2009 Annual Computer Security Applications Conference, ACSAC '09, Washington, DC, USA, pp.340–349, IEEE Computer Society, 2009.

[110] W. Enck, P. Gilbert, B.G. Chun, L.P. Cox, J. Jung, P. McDaniel, and A.N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, 2010.

[111] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.R. Sadeghi, and B. Shastry, "Towards taming privilege-escalation attacks on android," 19th Annual Network and Distributed System Security Symposium (NDSS), 2012.

[112] S. Smalley and R. Craig, "Security Enhanced (SE) Android: Bringing Flexible MAC to Android," NDSS, The Internet Society, 2013.

[113] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D.S. Wallach, "Quire: Lightweight provenance for smart phone operating systems," 20th USENIX Security Symposium, 2011.

[114] N. Good, R. Dhamija, J. Grossklags, D. Thaw, S. Aronowitz, D. Mulligan, and J. Konstan, "Stopping spyware at the gate: A user study of privacy, notice and

spyware," Symposium on Usable Privacy and Security (SOUPS), pp.43–52, 2005.

[115] C. Bravo-Lillo, L.F. Cranor, J. Downs, and S. Komanduri, "Bridging the gap in computer security warnings: A mental model approach," IEEE Security & Privacy, vol.9, no.2, pp.18–26, 2011.

[116] D. Akhawe and A.P. Felt, "Alice in warningland: A large-scale field study of browser security warning effectiveness," Proceedings of the 22Nd USENIX Conference on Security, Security'13, 2013.

[117] W. Chen, D. Aspinall, A.D. Gordon, C. Sutton, and I. Muttik, "A text-mining approach to explain unwanted behaviours," Proceedings of the 9th European Workshop on System Security, p.4, ACM, 2016.

[118] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for android apps," Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp.518–529, ACM, 2015.

List of Research Achievements

Journal Papers

1. Takuya Watanabe, Mitsuaki Akiyama, Fumihiro Kanei, Eitaro Shioji, Yuta Takata, Bo Sun, Yuta Ishii, Toshiki Shibahara, Takeshi Yagi, and Tatsuya Mori, "Study on the Vulnerabilities of Free and Paid Mobile Apps Associated with Software Library," IEICE Transactions on Information and Systems, vol.E103-D, no.2, to appear, February 2020.
2. Takuya Watanabe, Eitaro Shioji, Mitsuaki Akiyama, Keito Sasaoka, Takeshi Yagi, and Tatsuya Mori, "Follow Your Silhouette: Identifying the Social Account of Website Visitors through User-Blocking Side Channel," IEICE Transactions on Information and Systems, vol.E103-D, no.2, to appear, February 2020.
3. Takuya Watanabe, Mitsuaki Akiyama, Tetsuya Sakai, Hironori Washizaki, and Tatsuya Mori, "Understanding the Inconsistency between Behaviors and Descriptions of Mobile Apps," IEICE Transactions on Information and Systems, vol.E101-D, no.11, pp.2584–2599, November 2018.
4. Takuya Watanabe, Mitsuaki Akiyama, and Tatsuya Mori, "Tracking the Human Mobility Using Mobile Device Sensors," IEICE Transactions on Information and Systems, vol.E100-D, no.8, pp.1680–1690, August 2017.
5. Bo Sun, Xiapu Luo, Mitsuaki Akiyama, Takuya Watanabe, and Tatsuya Mori, "PADetective: A Systematic Approach to Automate Detection of Promotional Attackers in Mobile App Store," Journal of Information Processing, vol.26, pp.212-223, January 2018.
6. Yuta Ishii, Takuya Watanabe, Mitsuaki Akiyama, and Tatsuya Mori, "AP-

Praiser: A Large Scale Analysis of Android Clone App," IEICE Transactions on Information and Systems, vol.E100-D, no.8, pp.1703–1713, August 2017.

Conference Papers

1. Takuya Watanabe, Eitaro Shioji, Mitsuaki Akiyama, and Tatsuya Mori, "Melting Pot of Origins: Compromising the Intermediary Web Services that Rehost Websites," Proceedings of the Network and Distributed System Security Symposium (NDSS 2020), to appear, February 2020.
2. Takuya Watanabe, Eitaro Shioji, Mitsuaki Akiyama, Keito Sasaoka, Takeshi Yagi, and Tatsuya Mori, "User Blocking Considered Harmful? An Attacker-controllable Side Channel to Identify Social Accounts," Proceedings of IEEE European Symposium on Security and Privacy (EuroS&P 2018), pp.323–337, April 2018.
3. Takuya Watanabe, Mitsuaki Akiyama, Fumihiro Kanei, Eitaro Shioji, Yuta Takata, Bo Sun, Yuta Ishii, Toshiki Shibahara, Takeshi Yagi, and Tatsuya Mori, "Understanding the Origins of Mobile App Vulnerabilities: A Large-scale Measurement Study of Free and Paid Apps," Proceedings of International Conference on Mining Software Repositories (MSR 2017), pp.14–24, May 2017.
4. Takuya Watanabe, Mitsuaki Akiyama, and Tatsuya Mori, "RouteDetector: Sensor-based Positioning System That Exploits Spatio-Temporal Regularity of Human Mobility," Proceedings of USENIX Workshop on Offensive Technologies (WOOT 15), pp.1–11, August 2015.
5. Takuya Watanabe, Mitsuaki Akiyama, Tetsuya Sakai, Hironori Washizaki, and Tatsuya Mori, "Understanding the Inconsistencies between Text Descriptions and the Use of Privacy-sensitive Resources of Mobile Apps," Proceedings of Symposium On Usable Privacy and Security (SOUPS 2015), pp.241–255, July 2015.
6. Ayako Akiyama Hasegawa, Takuya Watanabe, Eitaro Shioji, and Mitsuaki Akiyama, "I Know What You Did Last Login: Inconsistent Messages Tell

Existence of a Target's Account to Insiders," Proceedings of Annual Computer Security Applications Conference (ACSAC 2019), pp.XX–XX, December 2019.

7. Asuka Nakajima, Takuya Watanabe, Eitaro Shioji, Mitsuaki Akiyama, and Maverick Woo, "A Pilot Study on Consumer IoT Device Vulnerability Disclosure and Patch Release in Japan and the United States," Proceedings of ACM ASIA Conference on Information, Computer and Communications Security (ASIACCS 2019), pp.485–492, July 2019.
8. Keika Mori, Takuya Watanabe, Yunao Zhou, Ayako Hasegawa, Mitsuaki Akiyama, and Tatsuya Mori, "Comparative Analysis of Three Language Spheres: Are Linguistic and Cultural Differences Reflected in Password Selection Habits?," Proceedings of Proceedings of the 4th IEEE European Workshop on Usable Security (EuroUSEC 2019), pp.159–171, June 2019.
9. Tatsuhiko Yasumatsu, Takuya Watanabe, Fumihiro Kanei, Eitaro Shioji, Mitsuaki Akiyama, and Tatsuya Mori, "Understanding the Responsiveness of Mobile App Developers to Software Library Updates," Proceedings of ACM Conference on Data and Application Security and Privacy (CODASPY 2019), pp.13–24, March 2019.
10. Yuta Ishii, Takuya Watanabe, Fumihiro Kanei, Yuta Takata, Eitaro Shioji, Mitsuaki Akiyama, Takeshi Yagi, Bo Sun, and Tatsuya Mori,, "Understanding the Security Management of Global Third-Party Android Marketplaces," Proceedings of International Workshop on App Market Analytics (WAMA 2017), pp.12–18, September 2017.
11. Bo Sun, Xiapu Luo, Mitsuaki Akiyama, Takuya Watanabe, and Tatsuya Mori , "Characterizing Promotional Attacks in Mobile App Store," Proceedings of International Conference on Applications and Techniques in Information Security (ATIS2017), pp.113–127, July 2017, **Best Paper Award**.
12. Yuta Ishii, Takuya Watanabe, Mitsuaki Akiyama, and Tatsuya Mori,, "Clone or Relative?: Understanding the Origins of Similar Android Apps," Proceedings of ACM International Workshop on Security And Privacy Analytics (IWSPA 2016), pp.25–32, March 2016.

Posters

1. Takuya Watanabe and Tatsuya Mori, "Understanding the Consistency Between Words and Actions for Android Apps," ACM ASIA Conference on Information, Computer and Communications Security (ASIACCS 2014), pp.Poster Session, June 2014, **Best Poster Award**.
2. Atsuko Natatsuka, Ryo Iijima, Takuya Watanabe, Mitsuaki Akiyama, Tetsuya Sakai, and Tatsuya Mori, "A First Look at the Privacy Risks of Voice Assistant Apps," ACM Conference on Computer and Communications Security (CCS 2019), pp.Poster Session, November 2019.
3. Bo Sun, Takuya Watanabe, Mitsuaki Akiyama, and Tatsuya Mori, "Seeing is Believing? The Analysis of Unusual Ratings and Reviews on Android App Store," International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2015), pp.Poster Session, November 2015.
4. Yuta Ishii, Takuya Watanabe, Mitsuaki Akiyama, and Tatsuya Mori, "Understanding the Origins of Similar Android Apps," International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2015), pp.Poster Session, November 2015.

Talks

1. Takuya Watanabe, "[Invited Talk] トップカンファレンス採録への道," セキュリティサマーサミット 2019, July 2019.
2. Takuya Watanabe, "[Refereed Talk] I Block You Because I Love You: Social Account Identification Attack Against a Website Visitor," Black Hat Europe, December 2018.
3. Takuya Watanabe, "[Invited Talk] Silhouette: Controlling Side Channel to Identify Social Account of Website Visitor," International Workshop on Security (IWSEC 2018), September 2018.
4. Takuya Watanabe, "[Invited Talk] Understanding the Inconsistencies between Text Descriptions and the Use of Privacy-sensitive Resources of Mobile

Apps," Workshop on Psychological Factors of Cybersecurity, April 2017.

5. Takuya Watanabe, "[Invited Talk] ACODE: Analyzing the Inconsistencies between User Expectations and the Developer Intentions of Mobile Apps," International Workshop on Security (IWSEC 2015) , August 2015.

Others

1. 渡邉卓弥, "新たなプライバシ脅威「Silhouette」の発見と対策への取り組み," NTT 技術ジャーナル Vol. 31 No. 2, pp.15-18, February 2019.
2. Takuya Watanabe, "Discovery of Silhouette–a New Threat to Privacy–and Our Efforts to Counter It," NTT Technical Review, Vol. 17 No. 3, pp.11-15, March 2019.
3. 渡邉卓弥，塩治榮太朗，秋山満昭，笹岡京斗，八木毅，森達哉, "ユーザブロック機能の光と陰: ソーシャルアカウントを特定するサイドチャネルの構成," コンピュータセキュリティシンポジウム 2017 論文集, pp.858–865, October 2017, **情報処理学会 CSS 最優秀論文賞, 山下記念研究賞**.
4. 渡邉卓弥，秋山満昭，森達哉, "RouteDetector: 9 軸センサ情報を用いた位置情報追跡攻撃," コンピュータセキュリティシンポジウム 2015 論文集, pp.1127–1134, October 2015, **情報処理学会 PWS 優秀論文賞**.
5. 渡邉卓弥，秋山満昭，森達哉, "Android アプリの説明文とプライバシー情報アクセスの相関分析," コンピュ ータセキュリティシンポジウム 2014 論文集, pp.590–597, October 2014, **情報処理学会 CSS 学生論文賞, MWS 学生論文賞**.
6. 渡邉卓弥，森達哉，酒井哲也, "カメラを秘密裏に濫用する Android アプリの検出," 電子情報通信学会技術研 究報告 (信学技報) Vol. 113 No. 502, pp.119–124, March 2014.
7. 刀塚敦子, 飯島涼, 渡邉卓弥, 秋山満昭, 酒井哲也, 森達哉, "Voice Assistant アプリの大規模実態調査," コンピュータセキュリティシンポジウム 2019 論文集, pp.618–625, October 2019, **情報処理学会 CSS 最優秀論文賞**.
8. 櫻井悠次, 渡邉卓弥, 奥田哲矢, 秋山満昭, 森達哉, "サーバ証明書解析によるフィッシングサイトの発見手法," コンピュータセキュリティシンポジウム 2019 論文集, pp.910–917, October 2019, **情報処理学会 CSS 学生論文賞**.

9. 森啓華, 長谷川 彩子, 渡邉卓弥, 笹崎寿貴, 秋山満昭, 森達哉, “パスワード生成アシスト技術の有効性評価:異なる言語圏のユーザを対象とした追試研究,” コンピュータセキュリティシンポジウム 2019 論文集, pp.214–221, October 2019, **情報処理学会 UWS 論文賞**.
10. 長谷川彩子, 渡邉卓弥, 塩治榮太朗, 秋山満昭, “ログイン関連画面に潜む脅威：センシティブサービスにおけるアカウント所有の特定,” コンピュータセキュリティシンポジウム 2019 論文集, pp.704–711, October 2019.
11. 高田雄太, 渡邉卓弥, 中野弘樹, 波戸邦夫, 秋山満昭, “Web プッシュ通知の悪用に関する実態調査,” 研究報 告コンピュータセキュリティ (CSEC) Vol. 2018 No. 17, pp.1–8, December 2018, **情報処理学会 CSEC 優秀研究賞**.
12. 櫻井悠次, 奥田哲矢, 秋山満昭, 渡邉卓弥, 高田雄太, 須賀祐治, 森達哉, “Web サイトのセキュリティ・センサス,” コンピュータセキュリティシンポジウム 2018 論文集, pp.77–84, October 2018.
13. 安松達彦，金井文宏，渡邉卓弥，塩治榮太朗，秋山満昭，森達哉, “モバイルアプリ開発者による脆弱性対応の実態調査,” コンピュータセキュリティシンポジウム 2017 論文集, pp.644–651, October 2017, **情報処理学会 MWS 学生論文賞**.
14. 石井悠太，渡邉卓弥，秋山満昭，森達哉, “Android クローンアプリの大規模分析,” コンピュータセキュリテ ィシンポジウム 2015 論文集, pp.207–214, October 2015, **情報処理学会 MWS 学生論文賞**.
15. 孫博，渡邉卓弥，秋山満昭，森達哉, “Android アプリストアにおける不自然なレーティング・レビューの解析,” コンピュータセキュリティシンポジウム 2015 論文集, pp.655–662, October 2015.
16. 石井悠太，渡邉卓弥，秋山満昭，森達哉, “正規アプリに類似した Android アプリの実態解明,” 研究報告コン ピュータセキュリティ (CSEC) Vol. 2014 No. 94, pp.187–192, March 2015, **電子情報通信学会 情報通信システムセキュリティ研究賞**.

Copyrights