5-2016

Exploring Privacy Leakage from the Resource Usage Patterns of Mobile Apps

Amin Rois Sinung Nugroho
University of Arkansas, Fayetteville

Exploring Privacy Leakage from the Resource Usage Patterns of Mobile Apps

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

by

Amin Rois Sinung Nugroho
Institute of Statistics
Bachelor of Applied Science in Computation Statistics, 2008

May 2016
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

Dr. Qinghua Li
Thesis Director

Dr. Xintao Wu
Committee Member

Dr. Tingxin Yan
Committee Member

Abstract

Due to the popularity of smart phones and mobile apps, a potential privacy risk with the usage of mobile apps is that, from the usage information of mobile apps (e.g., how many hours a user plays mobile games in each day), private information about a user's living habits and personal activities can be inferred. To assess this risk, this thesis answers the following research question: can the type of a mobile app (e.g., email, web browsing, mobile game, music streaming, etc.) used by a user be inferred from the resource (e.g., CPU, memory, network, etc.) usage patterns of the mobile app?

This thesis answers this question for two kinds of systems, a single mobile device and a mobile cloud computing system. First, two privacy attacks under the same framework are proposed based on supervised learning algorithms. Then these attacks are implemented and explored in a mobile device and in a cloud computing environment. Experimental evaluations show that the type of app can be inferred with high probability. In particular, the attacks achieve up to 100% accuracy on a mobile device, and 66.7% accuracy in the mobile cloud computing environment. This study shows that resource usage patterns of mobile apps can be used to infer the type of apps being used, and thus can cause privacy leakage if not protected.

Table of Contents

List of Figures

List of Tables

I. INTRODUCTION

A. Overview

With the popularity of smart mobile phones such as iPhone and Android phones, mobile apps installed on those phones play an increasingly important role in people's life, allowing users to check emails, play games, listen to music, do online banking, perform online social activities, and so on.

Despite the convenience provided by mobile apps, there is a potential privacy risk with the usage of mobile apps. In particular, from the usage information of mobile apps, private information about a user's living habits and personal activities might be inferred. For example, the use of game apps can reveal how much time a user spends on playing, and the use of music streaming apps can reveal whether a user is a music fan or not. To assess such privacy leakage risk, it is important to assess the risk of mobile app usage information being exposed to the attacker.

B. Research Questions

In this thesis, we want to answer the following research question: can the resource (e.g., CPU, memory, network, etc.) usage patterns of a mobile app be used to infer the type of mobile app (e.g., email, web browsing, mobile game, music streaming, etc.) being used by a user? We answer this question for two kinds of systems, a single mobile device and a mobile cloud computing system.

In particular, the first sub-question that this thesis answers is whether the type of an app can be inferred from the resource usage patterns of the app measured on mobile devices. Although there are some APIs provided by mobile operating systems (e.g., Android) through

which an attacker can get the app currently being used by a user, this attack can be mitigated by existing static analysis and dynamic analysis techniques [1].

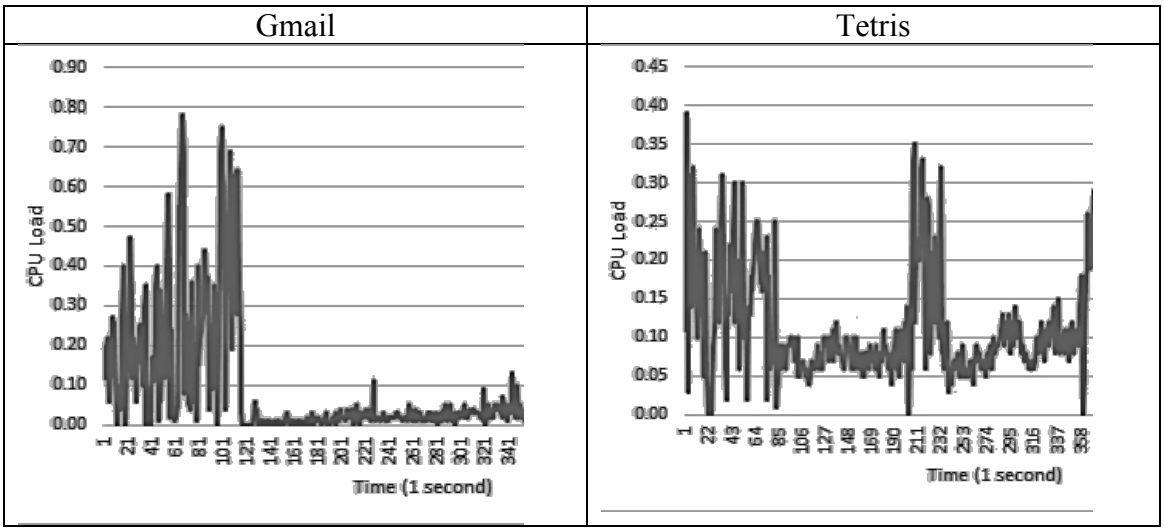

Figure 1. CPU Usage Pattern of Two Applications

Instead, we explore a side channel attack that exploits the resource usage patterns of mobile apps. We observe that different apps have different resource usage patterns. For example, games and email clients may request different rates of CPU cycles or different amounts of memory to perform their tasks. Figure 1 shows the CPU usage pattern of two apps Gmail and Tetris obtained on Asus Nexus 7 tablet running Android 5.1. Here Gmail is an email client and Tetris is a game. It can be seen that the CPU usage pattern of the two apps are very different. The maximum CPU load of Gmail is 0.80 (out of 1.0) while the maximum CPU load for Tetris is 0.40. Moreover, the minimum CPU load of Gmail is 0.00 while that of Tetris is 0.05. The CPU usage pattern for Gmail also has longer low-value segments which look almost flat. Such difference might be exploited by an attacker who is capable of getting the resource usage pattern of a specific app to identify the type of apps being used by a user.

The second sub-question that this thesis answers is whether the type of an app can be inferred from the resource usage patterns of the app measured on the cloud server in a mobile cloud computing environment. In mobile cloud computing, mobile phones offload some computation tasks to cloud servers that have more powerful computing resources than phones. Following the growth of mobile devices in the market, mobile cloud computing offerings have also grown rapidly. However, this growth has not achieved its expected potential due to security and privacy concerns. Cloud servers always know the resource usage of the offloaded computation because they are the provider of the computing resources. Thus, there is a possibility that the server identifies the type of app offloaded to it. Similar to an electric company that does not need to know how we use the electricity in our homes, cloud servers should never know what the users are doing on their cloud servers. This knowledge is part of users' privacy. Therefore, the potential privacy leakage resulting from exploiting resource usage patterns of offloaded computations should be studied. Unfortunately, this problem has not been addressed in the literature.

Note that for the same app, when it runs in a mobile phone and when it is offloaded to a cloud server, its resource usage patterns might be different due to the difference in computer architecture and hardware resources between the mobile phone and the cloud server. Thus, the two sub-questions should be answered separately.

C. Contributions

To answer these two questions, we propose an attack framework with two attack methods that exploit the resource usage pattern of mobile apps to identify the type of apps. We also explore the attacks on a dataset collected from mobile device and a dataset collected from a cloud server. The contribution of this thesis is summarized as follows:

1. This thesis explores whether the resource usage pattern of mobile apps can be used to infer the type of apps being used. It identifies a side channel attack against mobile user privacy.

2. This thesis investigates the feasibility of resource usage pattern-based privacy attacks on both mobile devices and mobile cloud computing servers, and finds that the type of app can be inferred with high probability. These results can deepen our understanding of privacy attacks on mobile systems and call for solutions for enhancing privacy in mobile phones and mobile cloud computing systems.

D. Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter II reviews related work. Chapter III describes our attack framework and the two attack methods. Chapter IV and Chapter V describe the attacks on a mobile device and in a mobile cloud computing system, respectively. Chapter VI concludes the paper and points out future work.

II. LITERATURE REVIEW

Felt et al. (2011) [5] have conducted a mobile malware survey. They concluded that mobile malwares are primarily caused by permissions misuse by the applications. They discussed security and privacy threats while mobile devices are connected to the Internet and misbehaving applications have been installed. These misbehaving applications will send sensitive information about their users to the developer of the applications without the users' knowledge. The applications' developers will use this information for their financial gain, such as selling these data to interested third parties. The solutions that are suggested are better regulation of publishing mobile applications to the market and warning users to use mobile antivirus application and to be more aware when installing new mobile applications. Although it seems like a good solution to serious threats, potential security and privacy risks with undiscovered side channels such as resource usage patterns and in cloud computing scenarios were not discussed.

Xu et al. (2015) [7] have worked on profiling mobile apps by monitoring network traffic. They collect http traces and try to identify the app that generates those traces by using a unique identifier (msid=X) in the HTTP header. After finding the unique identifier inside the HTTP header, the identifier will be promoted as a signature when the correlation with a particular app is high. Then using these apps signatures, they tested their implementation, FLOWR, on real world traffic data and apps running on emulator to find the ground truth. They compared both data to test the accuracy. FLOWR is capable of identifying 90% of apps without relying on the seed signatures [7]. Yao et al. (2015) [8] developed a framework as an improvement to FLOWR. This framework is tested on 15 million flows generated by over 700K applications from the Android,

iOS, and Nokia market-places and is able to identify over 90% of the applications with 99% accuracy on average.

Similarly, Dai et al. (2013) [23] have worked on identifying Android apps from network traces. They proved that it is feasible to identify mobile apps by looking for the key/value that corresponds to the app's name in network traces. Tongaonkar et al. (2013) [27] have done similar work using advertisement data flow ID on ad-supported apps. They proved that it is feasible to identify the app ID using the key/value found in the advertisement data flow. Moreover, their scheme is also able to successfully predict when a type of apps is used. For example, news apps are mostly used in the morning. Although these works have similar goals with our work in identifying an app and thus inferring what the user is doing, they do not consider CPU usage data which is the focus of our work and they do not consider mobile cloud computing scenarios.

Devarakonda et al. (1989) [25] have worked on predicting how much CPU time and how much memory a known program will use. The goal is to help load balancing schemes to perform better by predicting the resource usage of a known program. Along that line, Shimizu et al. (2009) [26] have done work to predict the CPU time of a known program to better distribute the program to hosts in a distributed computing environment. These works infer the CPU usage of a known program. In contrast, our work infers the type of a mobile app from the known CPU usage pattern.

In mobile cloud computing, mobile devices can depend on cloud computing resources and cloud storage resources to execute computationally exhaustive tasks such as data mining and multimedia processing. Mobile cloud computing has grown rapidly, and this growth trend is expected to continue. Khan et al. (2013) describe that mobile cloud subscribers growth is still

lower than what had been expected because of security and privacy concerns. Based on a recent survey by the International Data Corporation, most IT Executives and CEOs are not interested in implementing such services due to the risks associated with security and privacy [2]. Security threats have become a barrier to taking advantage of the quick flexibility of the mobile cloud-computing paradigm. Substantial work has been done in research organizations and academia to develop secure mobile cloud computing environments and infrastructures [2]. However, these initiatives have not fully addressed the privacy concerns with mobile cloud computing.

Chun et al. (2011) [3] have developed Clone Cloud, currently the most efficient mobile cloud offloading implementation. Chun et al. claimed that Clone Cloud is able to increase execution time up to 20 times while decreasing energy intake up to 20-fold. Despite this huge gain resulting from adopting mobile cloud technology, security and privacy concerns continue to prevent many organizations from adopting it. Therefore, it is essential to identify what has been done to mitigate these potential security and privacy risks. It is clear that these risks play a more decisive role than the substantial benefits to be gained by adopting mobile cloud computing. Moreover, analyzing any potential security and privacy risks that remain must be addressed. Ren et al. (2012) [4] have discussed various security challenges in public clouds such as data service outsourcing security, computation outsourcing security, trustworthy service metering, access control, multi-tenancy, security and privacy, and security overhead. Ren et al. admit that answering these challenges is key to successful cloud computing adoption. However, solutions to these challenges were not explained.

Huang and Zhou (2011) [6] have developed a secure framework for mobile cloud computing through trust management and private data isolation. The trust management part includes identity management, key management, and security policy enforcement. While this

solution seems to provide security assurance, it requires one public cloud server to be trusted in addition to requiring at least two public cloud providers. Therefore, this solution requires more resources to provide its security properties. Moreover, it is still vulnerable to other potential privacy leakage by exploiting resource usage patterns. Slamanig (2013) [24] proposed a scheme to mitigate privacy leakage in the cloud environment by hiding cloud user's actual resource usage using an upper bound limit and partially blind signed token. However, that scheme only considers how long the CPU is used instead of the real-time CPU load profiles.

To summarize, potential privacy leakage by exploiting resource usage patterns are among those unanswered concerns. This thesis aims to investigate the possibility of privacy leakage resulting from resource usage patterns.

III. ATTACK FRAMEWORK

This chapter describes the framework of resource usage pattern-based attack.

A. System Model and Basic Approach

We assume that the attacker can observe the resource usage time-series of mobile apps used by a user. In a mobile device platform, this can be done through installing a resource usage monitoring app (or malware) and/or exploiting system functions. For example, the attacker can install Android Debug Bridge [13] in its own remote computer. Due to support provided by the Android OS, this tool can remotely monitor the CPU and memory usage of apps on the mobile device through the network without installing any additional tool on the mobile device. In a mobile cloud computing system, the cloud server can be seen as the attacker, and it can observe the resource usage of mobile apps offloaded to it. The attacker can also be a third party that has gained access to the victim's resource usage data on the cloud server.

Given the resource usage data of one app, the attacker wants to learn the *type* of the app. Here type is defined as the category of apps, such as email client, web browser, game, music streaming tool, etc. In this thesis, we use type and category in an interchangeable way. There are two reasons why the attacker chooses to infer the type of an app instead of what specific app it is. First, since apps of the same type might have similar resource usage patterns due to similar user usage patterns, it is easier for the attacker to infer the type of an app than to infer which specific app it is. Second, although knowing what the app is gives more information about the user, knowing the type of the app can also tell much about the user's life pattern and living habits. For example, knowing that a user plays game for 8 hours a day is enough to conclude that the user is a game fun or even addicted to game, no matter which games he plays.

We also assume that the attacker knows the resource usage pattern of certain apps in each type. This is because the attacker can run those apps on his own mobile device or mobile cloud computing system and measure their resource usage time-series. Thus, the research problem is, with known resource usage time-series of certain apps whose type is also known, given a resource usage time-series of an unknown app, how to infer the type of the unknown app.

To address this problem, we adopt the supervised learning approach [9]. In supervised learning, some training data with labels will be used to build a classification model, and this model will be used to predict the label of testing data. In this thesis, each data sample is a resource usage time-series of an app, with t data points (i.e., t readings of resource usage at t continuous time points with constant intervals). Data label is the type of an app. Those known apps' types are known, i.e., their types are known labels. The known apps and their labels are the training data. The unknown app's resource usage time-series are testing data. The learning goal is to identify the type (i.e., label) of the unknown app. In particular, two attack methods are explored as described below.

B. Attack Method 1

In this method, the t data points of each time-series data sample are divided into a number of segments where each segment has w data points (w is a system parameter). Each segment of a training data sample inherits the label of this training data sample. For example, if a data sample in the training data is labeled as Gmail, then all the segments that it is divided into are also labeled as Gmail. All the segments of training data samples are input into a supervised learning algorithm to build the classification model. To learn the type of a testing data sample with t data points, it is also divided into t/w segments with w data points in each segment. Then each of

these segments is input to the classification model and is assigned a label after the classification. The label that is assigned most times is set as the label for the testing data sample.

Parameter *w* will affect the performance of this attack method. Intuitively, if *w* is too small, each segment is too short. Then it cannot capture the inherent fluctuation characteristics of an app's resource usage pattern that can only be observed in a long-enough time window, which will lead to poor attack performance. On the other hand, if *w* is too large, each segment might contain too many fluctuation cycles (e.g., CPU usage up and down) of an app's resource usage pattern, which will also lead to poor attack performance due to the coarse grain. Thus, there should be a good *w* in the middle that has the best performance. We will explore the best *w* in experiments.

There are many supervised learning algorithms that can be used in this method. In this thesis, we will explore k-nearest-neighbor, support vector machine, neural networks, classification tree, and Random Forests [11, 18]. In our experiments, Random Forests works best. Random Forest works by growing many classification trees. To classify a new object from an input vector, the input vector will be put in each of the tree in the forest. Then each tree will give a classification. In other words, each tree will vote for that class. The forest then decides the classification based on the class that has the most votes from all the trees.

C. Attack Method 2

This attack method employs the k-nearest-neighbor classification algorithm [16] and computes the distance between data samples using the Dynamic Time Warping (DTW) algorithm [17]. The idea is to find the *k* data samples in training data that are closest to the testing data sample, and then assign the label that most frequently appears among those k data samples as the label of the testing data sample. To determine how close two data samples are to

each other, the DTW algorithm is used to compute the distance between them. DTW can measure the similarity of time series data that are not best aligned and vary in time or speed. Then it will output the distance of two time series data after being aligned in the best way based on the similarity of those data. To improve the performance over classic DTW, FastDTW [12] is used.

IV. PRIVACY LEAKAGE FROM RESOURCE USAGE PATTERNS IN A MOBILE DEVICE

This section explores the proposed attacks on resource usage data collected from a mobile device.

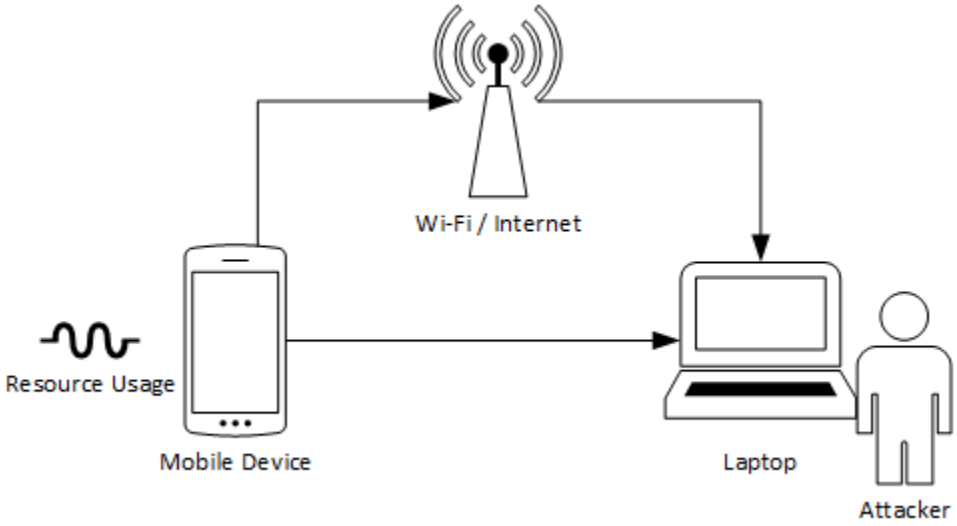

Figure 2. Overview of Attacks in a Mobile Device

A. Dataset collection

1) Overview

We first collect the resource usage patterns of mobile apps from a Nexus 7 tablet that runs the Android 5.0 operating system. Via USB port, the tablet is connected to a MacBook Pro laptop (see Figure 2). The laptop is installed with Android Debug Bridge [13]. Android Debug Bridge was developed and provided by Google to help Android developers monitor their apps' performances, including CPU usage and memory allocation. This tool was developed to help developers design more efficient applications. However, this capability is used in this work to monitor the CPU load and memory consumption of mobile apps. In particular, we use the top

command, which is available in Android Debug Bridge shell to read the resource usage of the Android device in every second. In addition to USB port connection, Android Debug Bridge can also be connected to any Android device via WiFi.

We collect resource usage data from five types of apps. They are the types that are widely used by users. Then we choose four most popular apps for each type according to Google Play Store. It is more reasonable to test widely used applications to reflect real life scenarios. The types and apps in each type are shown in Table I.

TABLE I. LIST OF APPS MEASURED

Type	Apps
Email Client	Gmail, Cloud Magic, Mailbox, Boxer
Games	Fruit Ninja, Tetris Blitz, Chess, Bejeweled Blitz
Cloud Storage	Box, Google Drive, Dropbox, One Drive
Web Browser	Chrome, Opera, Dolphin, Firefox
Music Streaming Service	Pandora, Spotify, Rdio, Google Music

We run each app for 10 minutes and collect its resource usage data. When running an app, the app performs a set of activities allowed by this app just like a regular user. Details of activities performed for each type of app can be seen in Table II.

TABLE II. ACTIVITIES FOR EACH TYPE OF APP

Type	Activities
Email Client	Checking email, reading email, typing email, sending email, replying email, and deleting email. Checking and changing some available setting.
Games	Playing several levels until timer expires.
Cloud Storage	Checking contents, opening file, create folder, uploading and downloading several files (PDF, image, text, video), copying and moving files between folder, and editing text files.
Web Browser	Visiting several usually visited websites, such as www.apple.com, www.uark.edu, www.theverge.com, and others.
Music Streaming Service	Playing songs from Taylor Swift radio station and tapping thumb-up or thumb-down to rate the songs.

The resources measured include CPU load (i.e., the percentage of CPU time spent in the measured app) and the total memory allocated (i.e., the amount of memory in megabytes allocated to the measured app). They are read once per second.

To verify the correctness of the collected data, a video of the mobile device screen will be recorded in the experiment. This video is used to confirm a causal connection between the resource reading and the task that is actually being performed by an app. For example, when the send button is pressed in an email client, it should incur higher measured CPU usage than when the text in the email client is being typed, since typing the text in an email client requires less CPU cycles than sending the email to the network. For another example, a web browser will do more computations while downloading and rendering the web pages than when the user is just reading the web pages. We manually check the consistency between user operations and the measured resource usage to make sure the collected data is correct.

2) Data Collection and Processing

The detailed steps of our data collection method are as follows:

1. After connecting the Nexus 7 Tablet to the MacBook Pro laptop via USB port, open the Terminal app on laptop, and then launch the following top command to record the resource usage. Option –m is used to show memory usage and option –d is used to set the delay between readings.

   /Applications/sdk/platform-tools/adb shell top -m 10 -d 01.00 > chrome.txt

2. The output of the above command is saved in the chrome.txt file, and one instance of the output is shown in Figure 3. This data contains the resource usage of all running apps in one second. The CPU usage, in percentage, can be seen in the column labeled as "CPU%". The amount of memory used, in kilobyte, can be seen in the column labeled as

"RSS". The amount of memory requested, in kilobyte, can be seen in the column labeled as "VSS". The name of application can be seen in the Name column.

```
User 11%, System 12%, IOW 1%, IRQ 0%
User 30 + Nice 3 + Sys 37 + Idle 214 + IOW 3 + IRQ 0 + SIRQ 1 = 288

  PID PR CPU% S  #THR     VSS     RSS PCY UID      Name
17843  3  10% S    37 1246364K  43580K  fg u0_a80   com.teslacoilsw.launcher
31863  0   7% S    92 1331644K  77248K  fg system   system_server
 8301  0   5% R    13 1211744K  34252K  fg u0_a35   com.android.chrome
  119  0   3% S    16  26772K   4604K  fg system   /system/bin/surfaceflinger
 8608  3   2% R     1   2492K    980K  fg shell    top
   82  0   1% R     1      0K      0K  fg root     mmcqd/0
31628  3   0% S    11  37320K   5168K  fg media    /system/bin/mediaserver
 8235  0   0% S     1      0K      0K  fg root     kworker/0:3
   94  3   0% S     1      0K      0K  fg root     dhd_dpc
 8493  3   0% S     1      0K      0K  fg root     irq/214-host_sp
```

Figure 3. A Snapshot of Resource Usage of All Applications

3. Then the resource usage of a single app (say, Chrome) can be obtained by selecting relevant lines of the raw data with the following command:

   cat chrome.txt | grep com.android.chrome > chrome_clean.txt

   The results are shown in Figure 4. They represent resource usage of the same app at different seconds. Each line in this data represents the resource usage of the app in one second, except the ones that show different process IDs (highlighted lines in Figure 4) which will be aggregated in Step 4.

```
8301  0   5% R    13 1211744K  34252K  fg u0_a35   com.android.chrome
8301  3  19% D    18 1257416K  44104K  fg u0_a35   com.android.chrome
8301  3  21% S    40 1327644K  83588K  fg u0_a35   com.android.chrome
8301  3   4% D    42 1301868K  58420K  fg u0_a35   com.android.chrome
8656  3   1% S    16 1261428K  33608K  fg u0_a35   com.android.chrome:privileged_proce
8301  0  30% R    47 1308848K  66164K  fg u0_a35   com.android.chrome
8624  3   5% S    17 1297992K  35936K  fg u0_i1    com.android.chrome:sandboxed_proces
8656  3   1% S    16 1267476K  35976K  fg u0_a35   com.android.chrome:privileged_proce
8301  3  31% S    55 1359632K  79572K  fg u0_a35   com.android.chrome
8624  3  12% S    19 1347388K  47392K  fg u0_i1    com.android.chrome:sandboxed_proces
8301  2  18% S    55 1371652K  92172K  fg u0_a35   com.android.chrome
8624  3   7% S    20 1348124K  48288K  fg u0_i1    com.android.chrome:sandboxed_proces
8656  3   0% S    16 1267976K  36796K  fg u0_a35   com.android.chrome:privileged_proce
```

Figure 4. CPU and Memory Usage for an App (Google Chrome)

4. Aggregation of multiple-process usage

   The highlighted lines in Figure 4 have different process ID. They represent resource usage of different processes that belong to the same app at the same second. Therefore, they need to be aggregated as single data point. This phenomenon mostly happens for web browser when the user opens several tabs at the same time and other applications that utilize OS services. We import these data into Microsoft Excel and then use VBA Script to aggregate them. The list of applications that utilize multiple processes can be seen in Table III. The list of applications that utilize multiple processes can be seen in Table III.

TABLE III. LIST OF APPLICATIONS WITH MULTIPLE PROCESSES

Type	Applications
Email Client	Gmail
Games	-
Cloud Storage	Google Drive, Dropbox
Web Browser	Chrome, Opera, Dolphin, Firefox
Music Streaming Service	Rdio, Google Music

5. We add one column on the left and fill it with series of number in second that will be the time stamp. Then the final raw data is derived. Figure 5 shows an example.

	A	B	C	D	E	F
1	timestamp	CPU Usage	Memory Usage		Type	App Name
2	1	0.07	37680		games	chess
3	2	0.05	47284		games	chess
4	3	0.08	58800		games	chess
5	4	0.13	74596		games	chess
6	5	0.17	82452		games	chess
7	6	0.11	82232		games	chess
8	7	0.10	82708		games	chess
9	8	0.07	83204		games	chess
10	9	0.11	88668		games	chess
11	10	0.10	88668		games	chess

Figure 5. Final Resource Usage Data for an App (Chess)

3) Dataset

The collected data can be drawn as graphs with time (in seconds) as its x-axis and resource usage as the y-axis to visualize the pattern. The CPU usage patterns of 20 apps are shown in Figure 6 and 7. It can be seen that apps from the same type have similar shapes of CPU usage patterns. On the contrary, apps from different types show different shapes of CPU usage patterns.

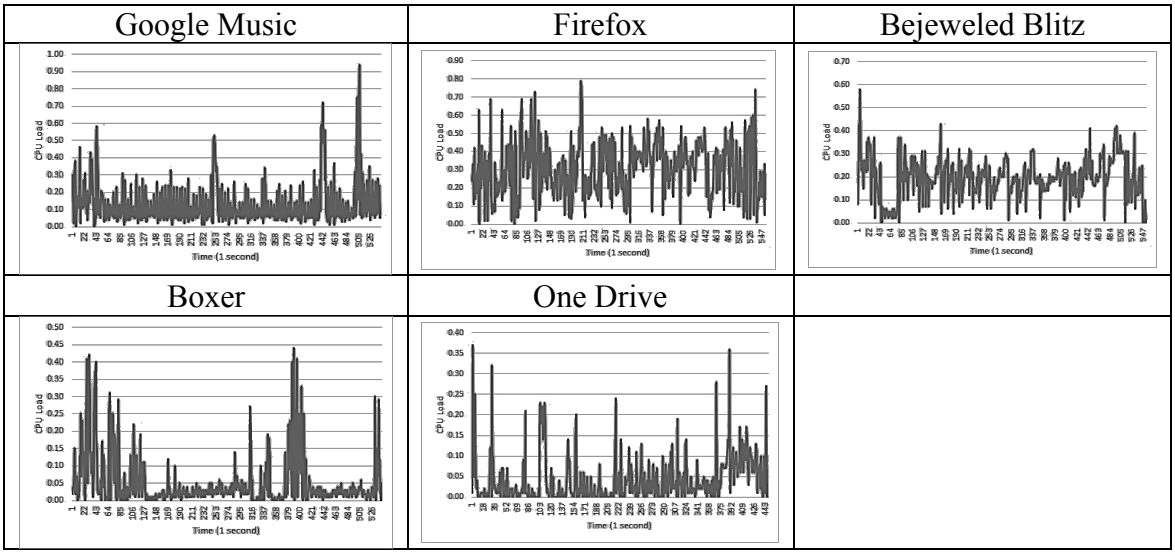

Figure 6. 10-minute CPU Usage Patterns of 5 Apps on Mobile Device

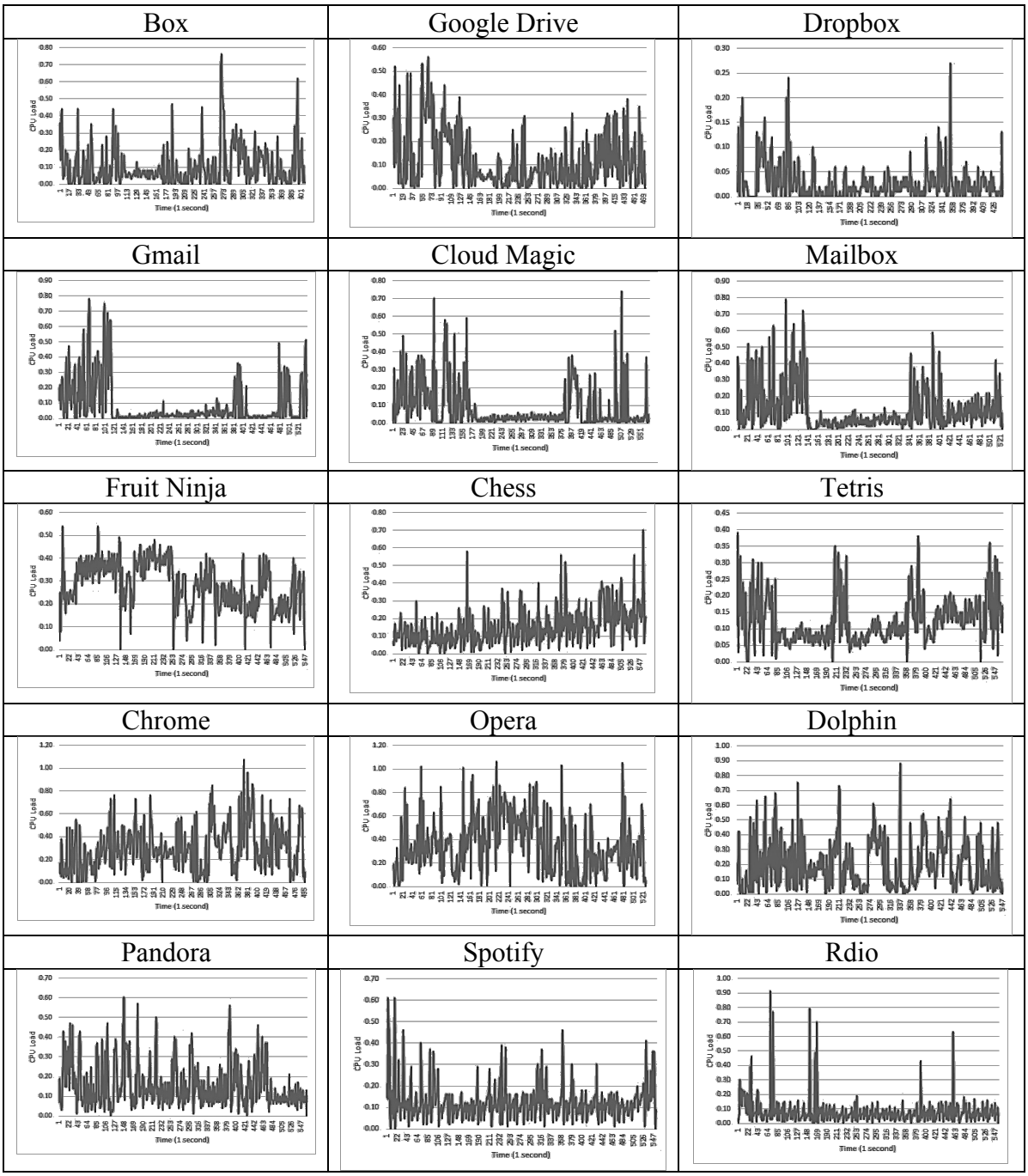

Figure 7. 10-minute CPU Usage Patterns of 15 Apps on Mobile Device

B. Results of Attack Method 1

Three applications from each type will be used as training data to build the classification model. One application from each type will be used as testing data to test the accuracy of the model. The apps in training data and testing data are shown in Table IV.

TABLE IV. LIST OF APPS FOR TRAINING AND TESTING

Category	**Applications**	
	Training Data	***Testing Data***
Email Client	Gmail, Cloud Magic, Mailbox	Boxer
Games	Fruit Ninja, Tetris Blitz, Chess	Bejeweled Blitz
Cloud Storage	Box, Google Drive, Dropbox	One Drive
Web Browser	Chrome, Opera, Dolphin	Firefox
Music Streaming Service	Pandora, Spotify, Rdio	Google Music

TABLE V. ACCURACY OF ATTACK METHOD 1 WITH DIFFERENT CLASSIFICATION ALGORITHMS ON MOBILE DEVICE USAGE DATA WHEN W=25 SECONDS

App (true type)	**Classification Results: Type (percentage of segments mapped to this type)**				
	Random Forest	**Neural Network**	**Nearest Neighbor**	**Support Vector Machine**	**Classification Tree**
One Drive (Cloud Storage)	Cloud Storage (66.67%)	Cloud Storage (0%)	Cloud storage (11.77%)	Cloud storage (0%)	Cloud storage (29.41%)
Boxer (Email Client)	Email Client (59.09%)	Email Client (95.2%)	Email client (61.90%)	Email client (95.24%)	Email client (47.62%)
Bejeweled Blitz (Games)	Games (91.30%)	Games (81.8%)	Games (81.82%)	Games (59.09%)	Games (68.18%)
Google Music (Music Streaming)	Music streaming (68.18%)	Music streaming (0%)	Music streaming (61.90%)	Music streaming (33.33%)	Music streaming (23.81%)
Firefox (Web Browser)	Web Browser (60.87%)	Web Browser (81.8%)	Web Browser (9.09%)	Web Browser (77.27%)	Web Browser (59.09%)
Accuracy	5/5 = 100%	3/5 = 60%	3/5 = 60%	3/5 = 60%	2/5 = 40%

TABLE VI. ACCURACY OF ATTACK METHOD 1 WITH DIFFERENT CLASSIFICATION ALGORITHMS ON MOBILE DEVICE USAGE DATA WHEN W=40 SECONDS

App (true type)	Classification Results: Type (percentage of segments mapped to this type)				
	Random Forest	**Neural Network**	**Nearest Neighbor**	**Support Vector Machine**	**Classification Tree**
One Drive (Cloud Storage)	Cloud Storage (75.0%)	Cloud Storage (18.2%)	Cloud storage (45.45%)	Cloud storage (0%)	Cloud storage (45.45%)
Boxer (Email Client)	Email Client (78.6%)	Email Client (76.9%)	Email client (69.23%)	Email client (92.31%)	Email client (53.85%)
Bejeweled Blitz (Games)	Games (85.7%)	Games (69.2%)	Games (100%)	Games (84.62%)	Games (69.23%)
Google Music (Music Streaming)	Music streaming (71.43%)	Music streaming (23.1%)	Music streaming (53.85%)	Music streaming (23.08%)	Music streaming (38.46%)
Firefox (Web Browser)	Web Browser (85.7%)	Web Browser (46.2%)	Web Browser (15.38%)	Web Browser (61.54%)	Web Browser (46.15%)
Accuracy	5/5 = 100%	3/5 = 60%	3/5 = 60%	3/5 = 60%	2/5 = 40%

TABLE VII. ACCURACY OF ATTACK METHOD 1 WITH DIFFERENT CLASSIFICATION ALGORITHMS ON MOBILE DEVICE USAGE DATA WHEN W=50 SECONDS

App (true type)	**Classification Results: Type (percentage of segments mapped to this type)**				
	Random Forest	**Neural Network**	**Nearest Neighbor**	**Support Vector Machine**	**Classification Tree**
One Drive (Cloud Storage)	Cloud Storage (66.67%)	Cloud Storage (0%)	Cloud storage (25%)	Cloud storage (12.5%)	Cloud storage (62.5%)
Boxer (Email Client)	Email Client (45.45%)	Email Client (40%)	Email client (70%)	Email client (90%)	Email client (40%)
Bejeweled Blitz (Games)	Games (90.91%)	Games (18.2%)	Games (100%)	Games (81.82%)	Games (63.63%)
Google Music (Music Streaming)	Music streaming (45.45%)	Music streaming (80%)	Music streaming (40%)	Music streaming (60%)	Music streaming (30%)
Firefox (Web Browser)	Web Browser (66.67%)	Web Browser (36.4%)	Web Browser (9.09%)	Web Browser (54.54%)	Web Browser (36.36%)
Accuracy	3/5 = 60%	1/5 = 20%	2/5 = 40%	4/5 = 80%	2/5 = 40%

TABLE VIII. ACCURACY OF ATTACK METHOD 1 WITH DIFFERENT CLASSIFICATION ALGORITHMS ON MOBILE DEVICE USAGE DATA WHEN W=60 SECONDS

App (true type)	Classification Results: Type (percentage of segments mapped to this type)				
	Random Forest	**Neural Network**	**Nearest Neighbor**	**Support Vector Machine**	**Classification Tree**
One Drive (Cloud Storage)	Cloud Storage (87.5%)	Cloud Storage (14.3%)	Cloud storage (0%)	Cloud storage (14.28%)	Cloud storage (57.14%)
Boxer (Email Client)	Email Client (50%)	Email Client (11.1%)	Email client (66.67%)	Email client (77.78%)	Email client (44.44%)
Bejeweled Blitz (Games)	Games (80%)	Games (0%)	Games (100%)	Games (100%)	Games (22.22%)
Google Music (Music Streaming)	Music streaming (70%)	Music streaming (44.4%)	Music streaming (55.55%)	Music streaming (44.44%)	Music streaming (44.44%)
Firefox (Web Browser)	Web Browser (90%)	Web Browser (44.4%)	Web Browser (11.11%)	Web Browser (66.67%)	Web Browser (66.67%)
Accuracy	4/5 = 80%	0/5 = 0%	3/5 = 60%	3/5 = 60%	2/5 = 40%

From Table V-VIII we can see that Random Forest algorithm is superior to other algorithms. It gives better overall accuracy. Moreover, Random Forest's accuracy is distributed better across categories. On the contrary, Support Vector Machine accuracy can differ significantly between categories (0% for Cloud Storage and 92.3% for Email Client). Then we move forward with Random Forest by examining the performance using different values of time split window (w). As can be seen in Figure 8, w = 40 seconds is better than other values.

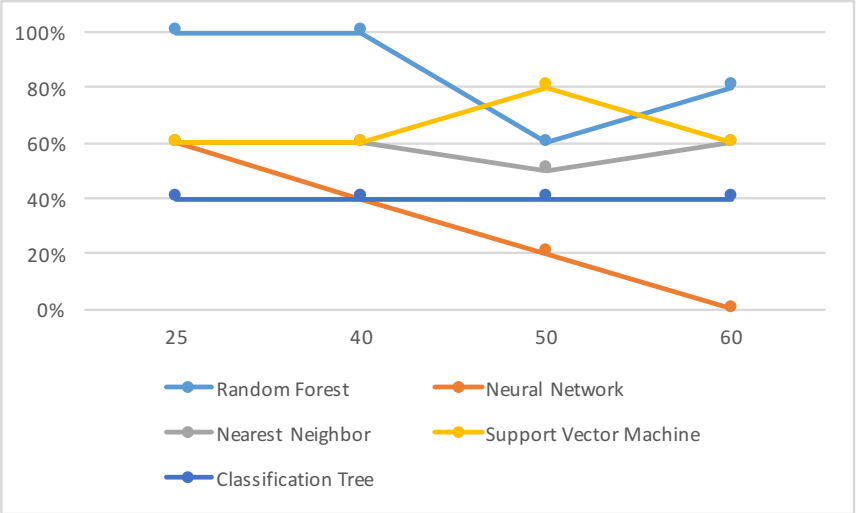

Figure 8. The accuracy of Attack Method 1 using different learning algorithms

C. Results of Attack Method 2

Then we tested Attack Method 2 that uses K-Nearest-Neighbor (KNN) with Dynamic Time Warping (DTW) distance since it is one of very good machine learning algorithm that works very well on time series data. The results are shown in Table IX. The accuracy with 3NN+DTW is 80%. We also tested 3NN+DTW on *w*-sized segments of data sample (similar to the Attack Method 1) with *w*=40, and the results are also shown in Table IX. The accuracy is the same as directly applying 3NN+DTW to the entire data samples.

TABLE IX. ACCURACY OF 3-NEAREST-NEIGHBORS WITH DYNAMIC TIME WARPING DISTANCE

Type	3-NN	3-NN on segments with 40 data points
Email Client	√	×
Games	√	√
Cloud Storage	×	√
Web Browser	√	√
Music Streaming Service	√	√
Accuracy	80%	80%

(√: Correct classification. ×: Wrong Classification)

To evaluate the effect of parameter *k*, we run experiments with different *k*. As can be observed in Figure 9, the best performance is achieved at *k*=3.

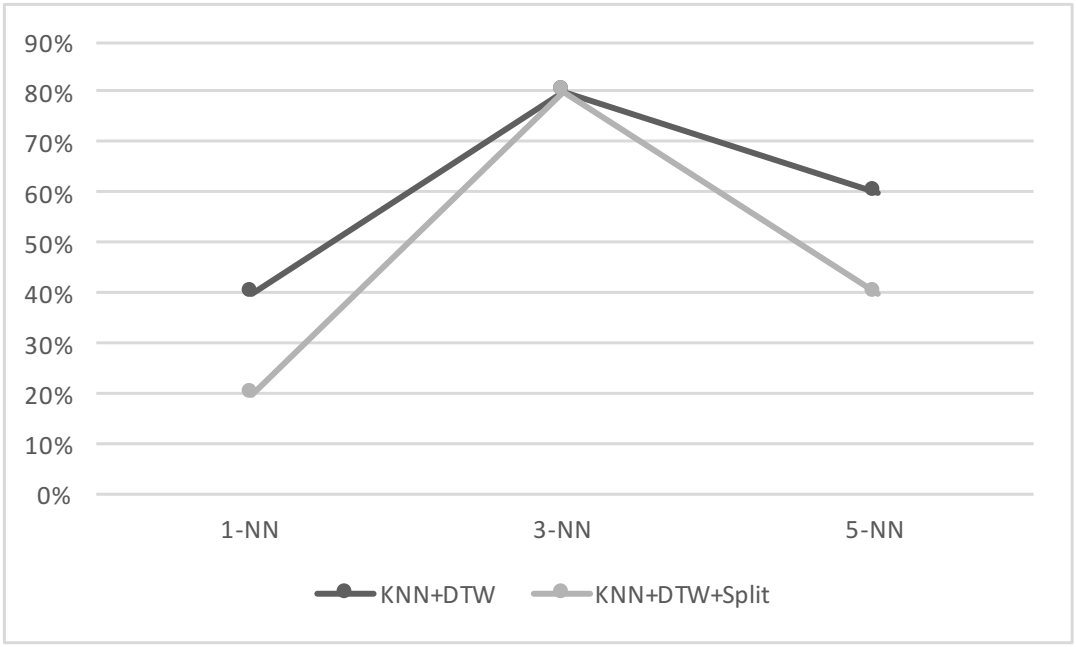

Figure 9. Accuracy of k-Nearest-Neighbors with Dynamic Time Warping Distance under Different k's

One factor that might affect the performance of attacks is the number of types (categories) and apps considered. Figure 10 shows the accuracy when the number of types changes. We first tested using four types of apps including email client, games, cloud storage, and web browser. Then we added music streaming service and maps for the fifth and sixth type of apps, respectively. The list of the apps can be seen in Table X.

It can be seen that when the number of types increases, the attack accuracy decreases. The reason is that Map apps have similar resource usage patterns with Web Browser apps. Generally speaking, when there are many types and apps, the accuracy should be good if there is clear distinction in the app functionality between apps' types. The more similarity between apps, the poor accuracy it will be. For example: messenger and email (both do sending and receiving messages), web browser and online maps (both do rendering of online pages from a server).

TABLE X. LIST OF APPS FOR TRAINING AND TESTING FOR FIGURE 10

Type	Apps	
	Training Data	***Testing Data***
Email Client	Gmail, Cloud Magic, Mailbox	Boxer
Games	Fruit Ninja, Tetris Blitz, Chess	Bejeweled Blitz
Cloud Storage	Box, Google Drive, Dropbox	One Drive
Web Browser	Chrome, Opera, Dolphin	Firefox
Music Streaming Service	Pandora, Spotify, Rdio	Google Music
Maps	Google Maps, Waze, Mapquest	Here Maps

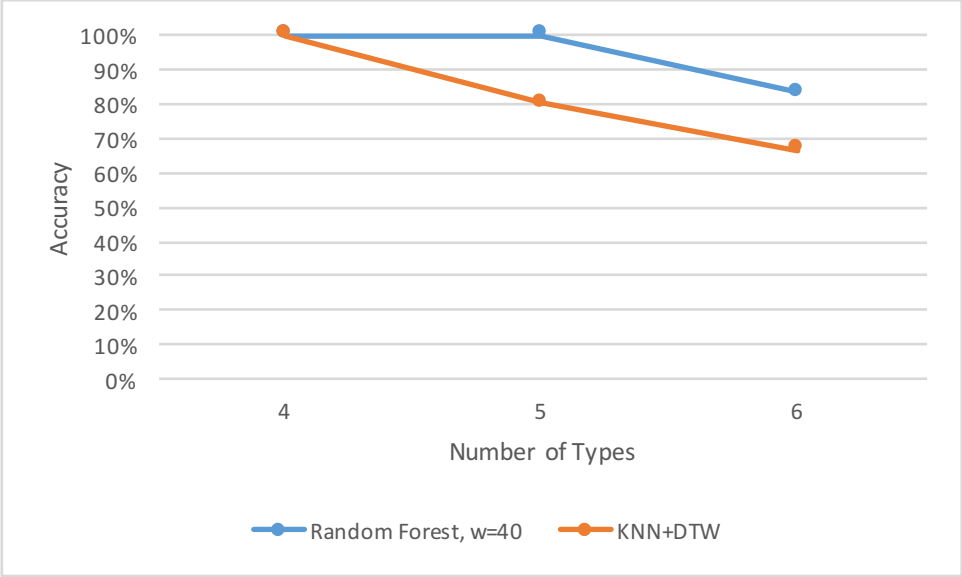

Figure 10. Accuracy of Attack Method 1 and Method 2 When the Number of Types Changes

V. PRIVACY LEAKAGE FROM RESOURCE USAGE PATTERNS IN MOBILE CLOUD COMPUTING

This section explores the proposed attacks in a mobile cloud computing environment.

A. Dataset Collection

Many mobile cloud computing approaches have been proposed [19, 20, 21, 22]. In a full offloading method, the whole app will be offloaded to the cloud server. This method does not require access to the app's source code. On the other hand, in partial offloading the app developers can mark which parts of their app will be offloaded. Developers have better knowledge on which parts of their applications need to be offloaded. However, this method requires access to the app's source code which we do not have. Therefore, in this work, we launch attacks to the full offloading method.

To set up the mobile cloud computing testbed using full offloading, we choose the COMET full offloading method [14]. COMET is chosen because it is one of the pioneer works in full offloading and its source code is made available by the authors [15]. COMET works by customizing the Android operating system to allow unmodified multi-threading apps to use more machines. It allows threads to migrate freely between the mobile device and the server depending on the workload. It also keeps enough information so that the mobile device can resume computation upon network failures. Figure 11 shows that the modified DalvikVM on the mobile device will communicate back and forth with the DalvikVM on the cloud server.

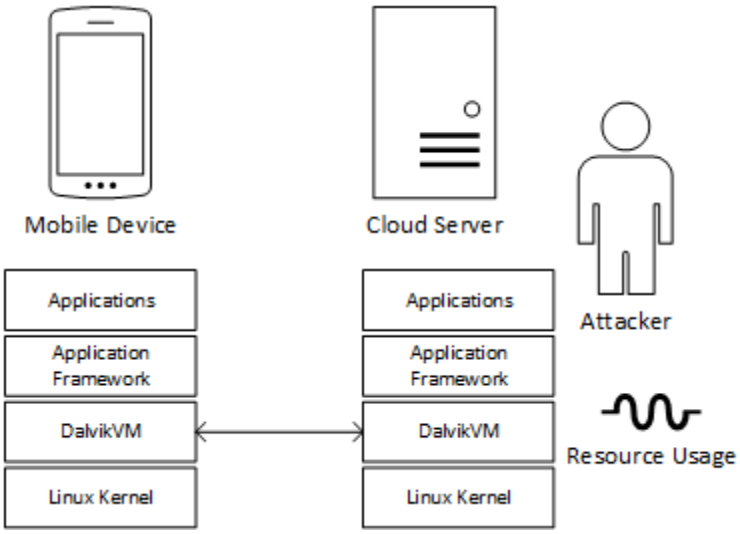

Figure 11. Overview of Attack on Mobile Cloud Using COMET.

Our testbed consists of one Nexus 7 tablet that serves as the mobile device (i.e., mobile cloud computing client) and one Linux-based laptop that serves as the cloud server. They are connected via WiFi. We first download and compile the COMET source code on the Linux-based laptop that satisfies the dependencies in compiling Android operating system source code. To support COMET, the CyanogenMod 10 operating system (a variant of Android) is installed on the Nexus 7 tablet. Then a modified version of DalvikVM, as a part of COMET, is installed to the Nexus 7 tablet, which is able to offload computations to the COMET tool in the laptop.

In the single mobile device experiments described in Chapter IV, we collected the resource usage data of 20 apps from five types (4 applications per type). However, at the time of doing experiments in the cloud computing environment, two apps have been shut down by the developer: Rdio, a music streaming service app and Mailbox, an email client. From those 18 remaining apps, only 9 apps from three types can run on the mobile cloud environment (3 apps per type). The offloading method is extending DalvikVM on multi-threading apps to work on another machine (cloud server/laptop). The apps that cannot run in cloud computing might not have been developed to support multi-threading.

Similar to the experiment on the mobile device, the data collection is done using the top command run on the cloud server. There are two processes associated with this offloading method that can be monitored on the cloud server: DalvikVM, the one responsible for computation on the cloud server/laptop and TCPmux, the one responsible for sending data from the mobile device/tablet to the cloud server/laptop. In the single mobile device environment, we can figure out the CPU usage of a specific application. On the contrary, in the mobile cloud environment, we can only monitor the total CPU usage since DalvikVM will be considered as a specific application by the Linux operating system in the cloud server.

B. Dataset

The CPU usage patterns of mobile apps measured in the cloud server are shown in Figure 12.

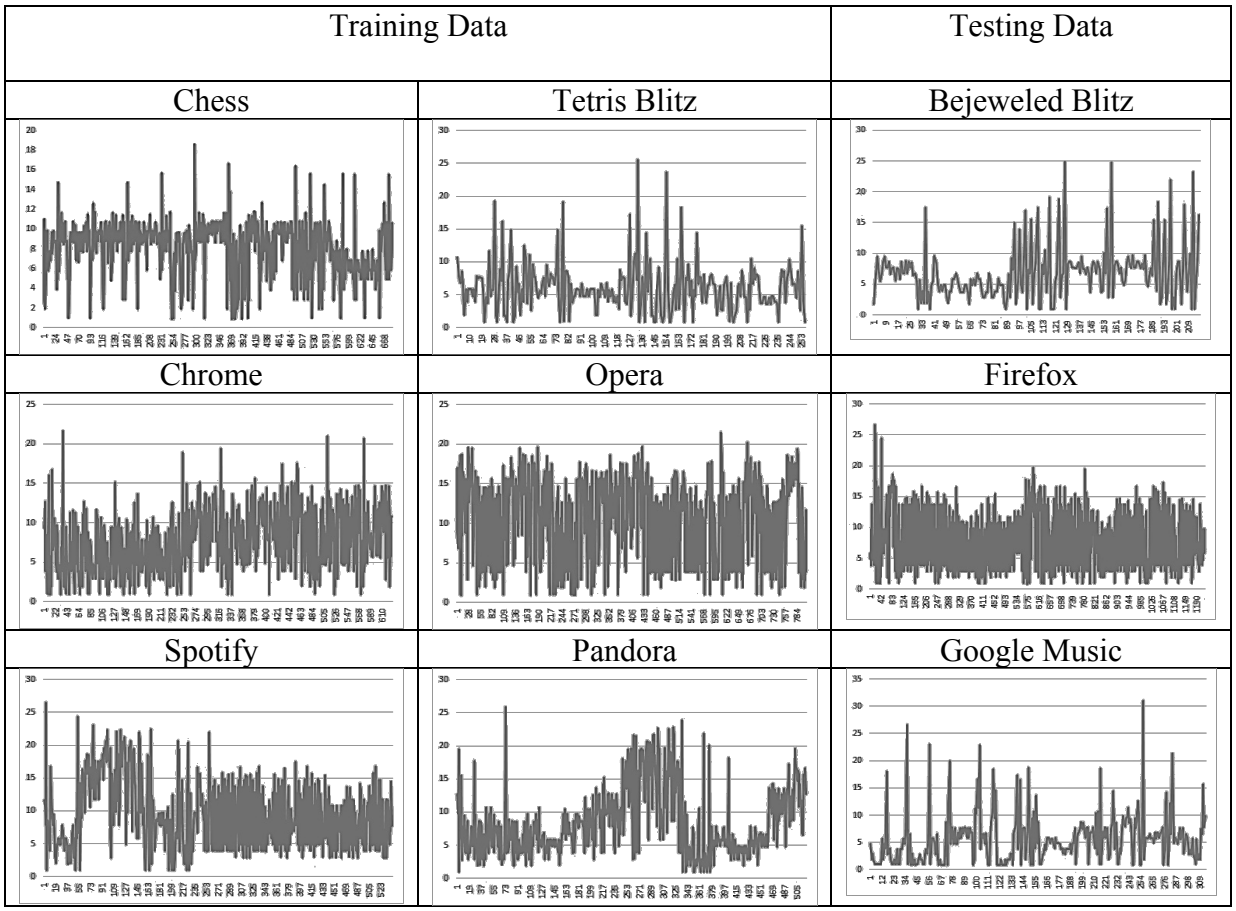

Figure 12. 10-minute CPU usage patterns in the mobile cloud computing environment

As mentioned before, for the same app, when it runs in a mobile device and when it is offloaded to a cloud server, its resource usage patterns might be different due to the difference in computer architecture and hardware resources between the mobile phone and the cloud server. Figure 13 shows the comparison of the CPU usage data of a particular app, Tetris, in the mobile cloud environment and in the mobile device. The pattern shapes of those data have some similarity. However, the CPU usage of Tetris when it is offloaded to a cloud server is lower. The maximum value of its CPU usage when offloaded is 0.25 (out of 1.0) compared to 0.40 (out of

1.0) when it runs on a mobile device. Thus, the effectiveness of our proposed attacks in mobile cloud settings should also be evaluated separately.

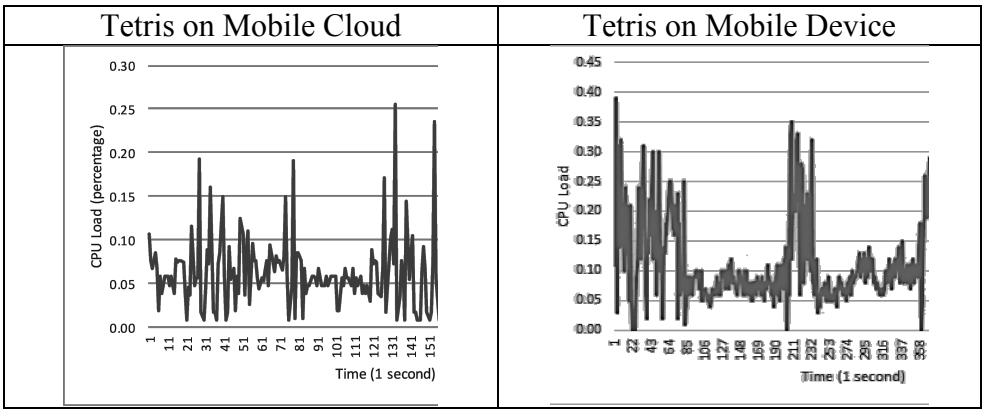

Figure 13. CPU Usage Pattern of an app Tetris on Two Platforms.

C. Results of Attacks

Using the CPU usage patterns of the DalvikVM in the cloud server, we evaluated the two learning algorithms that have the best performance in the mobile device based evaluation: KNN with DTW distance and Random Forest with 40-second data segments.

As we can see in Table XI and Table XII, Attack Method 2 performs better than Attack Method 1. Attack Method 2 can perform twice better than random guess accuracy which is 33.3%. On the other hand, Attack Method 1 only performs as good as random guess.

TABLE XI. ACCURACY OF ATTACK METHOD 1 WITH DIFFERENT CLASSIFICATION ALGORITHMS IN MOBILE CLOUD SERVER WHEN W=40 SECONDS

App (true type)	Classification Results: Type (percentage of segments mapped to this type)				
	Random Forest	**Neural Network**	**Nearest Neighbor**	**Support Vector Machine**	**Classification Tree**
Bejeweled Blitz (Games)	Games (83.33%)	Games (0%)	Games (80%)	Games (80%)	Games (60%)
Google Music (Music Streaming)	Music streaming (37.5%)	Music streaming (14.3%)	Music streaming (28.57%)	Music streaming (28.57%)	Music streaming (28.57%)
Firefox (Web Browser)	Web Browser (23.33%)	Web Browser (63.3%)	Web Browser (23.33%)	Web Browser (33.33%)	Web Browser (33.33%)
Accuracy	1/3 = 33.33%	1/3 = 33.33%	1/3 = 33.33%	1/3 = 33.33%	1/3 = 33.33%

TABLE XII. ACCURACY OF ATTACK METHOD 2 K-NEAREST NEIGHBORS WITH DYNAMIC TIME WARPING DISTANCE ON CLOUD BASED CPU USAGE DATA

Category	**1-NN**	**3-NN**
Games	√	√
Web Browser	√	√
Music Streaming Service	×	×
Accuracy	66.67%	66.67%

(√: Correct classification. ×: Wrong Classification)

VI. CONCLUSION AND FUTURE WORK

A. Conclusion

This thesis proposed privacy attacks that can infer the type of a mobile app from the resource usage patterns of this app. Through experiments based on datasets collected from a mobile device and a mobile cloud computing environment, we showed that the attacks can identify the type of mobile app from its resource usage patterns with high accuracy. In the single mobile device environment, the accuracy reaches 100%. In the mobile cloud computing environment, the accuracy reaches 66.7%. This study discovered that privacy leakage by exploiting resource usage patterns is technically feasible.

B. Future Work

One future work is to evaluate the attacks with more mobile apps and more types of mobile apps in both the mobile device and mobile cloud computing environments, and evaluate the attacks by many real-world users for a long time. Another direction is to evaluate the attacks for the partial offloading method of mobile cloud computing.

After evaluating these attacks, further research should be done to develop a countermeasure to mitigate these privacy leakages. One possible way would be to create additional noise in the resource usage patterns. This noise should be enough to make it improbable to distinguish the resource usage pattern of each application. However, it is also desirable to keep the cost needed to create this noise at a minimum.

REFERENCES

[1] W. Enck, L. P. Cox, P. Gilbert, and P. Mcdaniel, "TaintDroid : An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation, 2010, pp. 393–407.

[2] A. N. Khan, M. L. Mat Kiah, S. U. Khan, and S. a. Madani, "Towards secure mobile cloud computing: A survey," Futur. Gener. Comput. Syst., vol. 29, no. 5, pp. 1278–1299, Jul. 2013.

[3] B. Chun, S. Ihm, and P. Maniatis, "Clonecloud: elastic execution between mobile device and cloud," in EuroSys '11, 2011, pp. 301–314.

[4] K. Ren, C. Wang, and Q. Wang, "Security Challenges for the Public Cloud," Internet Comput. IEEE, vol. 16, pp. 69–73, 2012.

[5] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," Proc. 1st ACM Work. Secur. Priv. smartphones Mob. devices - SPSM '11, p. 3, 2011.

[6] D. Huang and Z. Zhou, "Secure data processing framework for mobile cloud computing," 2011 IEEE Conf. Comput. Commun. Work. (INFOCOM WKSHPS), pp. 614–618, Apr. 2011.

[7] Q. Xu, Y. Liao, S. Miskovic, Z. M. Mao, M. Baldi, A. Nucci, and T. Andrews, "Automatic Generation of Mobile App Signatures from Traffic Observations," in Computer Communications (INFOCOM), 2015 IEEE Conference on, 2015, pp. 1481–1489.

[8] H. Yao, "SAMPLES : Self Adaptive Mining of Persistent LExical Snippets for Classifying Mobile Application Traffic," in MobiCom '15 Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, 2015.

[9] M. Mohri, A. Rostamizadeh, A. Talwalkar, and M. Learning, Foundations of Machine Learning. Cambridge, MA: The MIT Press, 2012.

[10] MATLAB, http://www.mathworks.com/products/matlab/

[11] Random Forest, https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

[12] S. Salvador and P. Chan, "FastDTW : Toward Accurate Dynamic Time Warping in Linear Time and Space," Intell. Data Anal., vol. 11, pp. 561–580, 2007.

[13] Android Debug Bridge, http://developer.android.com/tools/help/adb.html

[14] M. Gordon, D. Jamshidi, and S. Mahlke, "COMET: code offload by migrating execution transparently," Proc. 10th …, pp. 93–106, 2012.

[15] Using Comet Source Code, http://www-personal.umich.edu/~msgsss/comet.html

[16] K. N. Stevens, T. M. Cover, and P. E. Hart, “Nearest neighbor pattern classification,” IEEE Trans. Inf. Theory, vol. 13, no. 1, pp. 21–27, 1967.

[17] S. Chiba, “Dynamic Programming Algorithm Optimization for Spoken Word Recognition,” IEEE Trans. Acoust., vol. 26, no. 1, pp. 43–49, 1978.

[18] L. Breiman, “Random Forests,” Mach. Learn., vol. 45, no. 1, pp. 5–32, 2001.

[19] H. Chen and Y. Lin, “COCA : Computation Offload to Clouds using AOP,” in 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012, pp. 466–473.

[20] Y. Li and W. Gao, “Code Offload with Least Context Migration in the Mobile Cloud,” in IEEE Conference on Computer Communications (INFOCOM), 2015, pp. 1876–1884.

[21] E. Cuervo, A. Balasubramanian, and D. Cho, “MAUI : Making Smartphones Last Longer with Code Offload,” in MobiSys’10, 2010.

[22] E. Y. Chen, S. Ogata, and K. Horikawa, “Offloading Android Applications to the Cloud without Customizing Android Invited Paper,” in Eighth IEEE PerCom Workshop on Pervasive Wireless Networking, 2012, no. March, pp. 788–793.

[23] S. Dai, A. Tongaonkar, X. Wang, A. Nucci and D. Song, "NetworkProfiler: Towards automatic fingerprinting of Android apps," IEEE Conference on Computer Communications (INFOCOM), 2013, pp. 809-817.

[24] D. Slamanig, “More privacy for cloud users: Privacy-preserving resource usage in the cloud,” Selected Papers from the 4th Hot Topics in Privacy Enhancing Technologies (HotPETs), 2011, pp. 15-27.

[25] M. V. Devarakonda and R. K. Iyer, "Predictability of process resource usage: a measurement-based study on UNIX," in IEEE Transactions on Software Engineering, vol. 15, no. 12, pp. 1579-1586, Dec 1989.

[26] S. Shimizu, R. Rangaswami, H. A. Duran-Limon, and M. Corona-Perez, “Platform-independent modeling and prediction of application resource usage characteristics,” J. Syst. Softw., vol. 82, no. 12, pp. 2117–2127, 2009.

[27] A. Tongaonkar, S. Dai, A. Nucci, and D. Song, “Understanding Mobile App Usage Patterns Using In-app Advertisements,” in Proceedings of the 14th International Conference on Passive and Active Measurement, 2013, pp. 63–72.