Inferring Mobile Apps from Resource Usage Patterns

Amin R. S. Nugroho
Dept. of Computer Science and Engineering
University of Arkansas
asnugroh@uark.edu

Qinghua Li
Dept. of Computer Science and Engineering
University of Arkansas
qinghual@uark.edu

***Abstract*—Despite many applications, mobile cloud computing induces privacy concerns. In particular, when mobile device users offload the computation of a mobile app to the cloud, they may not want the cloud service provider (CSP) to know what kind of app they are using, since that information might be used to infer their personal activities and living habits. One possible way for the CSP to learn the type of an offloaded app is to observe the resource usage patterns of the app (e.g., CPU and memory usage), since different apps have different resource needs due to their distinct computation workloads. To assess this risk, this paper answers the following question: Can the type of mobile app (e.g., email, web browsing, mobile game, etc.) used by a user be inferred from the resource usage pattern of the mobile app? We investigate the resource usage patterns of apps and whether the difference in resource usage pattern is sufficient to classify different types of apps. Specifically, two privacy attacks under the same framework are proposed based on supervised learning algorithms. Then these attacks are implemented and tested in a mobile device and in a cloud computing environment. Experiments show that, when the resource usage patterns on a mobile device are used, the type of app can be inferred with high probabilities; when the resource usage patterns on a cloud server are used, the type of app can be inferred with accuracy much higher than random guess.**

Keywords— Mobile cloud computing; privacy; mobile app; resource usage pattern; machine learning.

I. INTRODUCTION

With the popularity of smart mobile devices such as smartphones and tablets, mobile apps installed on those devices 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, etc. Due to the increasing number of apps running on mobile devices and the limited resources of mobile devices, mobile cloud computing has been proposed to offload expensive computations such as mathematical computing, data mining and multimedia processing to a cloud server so as to relieve the resource needs at mobile devices [3].

Despite the many applications and benefits of mobile cloud computing, there is a potential privacy risk with the offloading of mobile computations. Specifically, different apps may have different resource usage patterns such as CPU usage and memory usage. From the resource usage patterns of mobile apps, an attacker (e.g., an untrusted cloud server) might be able to infer the type of app being used by users, and in turn infer private information about users' personal activities and living habits. 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.

Figure 1 shows the CPU usage patterns 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 their CPU usage patterns are quite different. The maximum

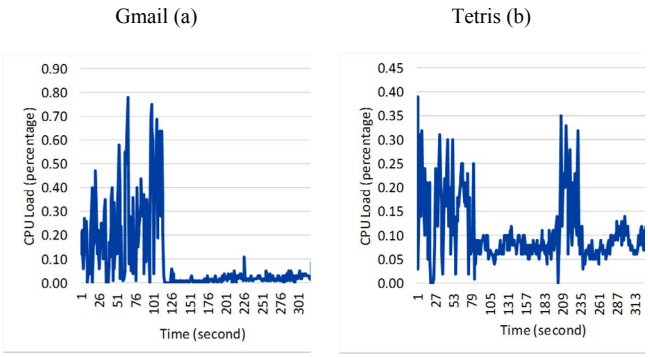

Fig. 1. The Different CPU Usage Patterns of Two Apps

CPU load of Gmail is 0.8 (out of 1.0) while the maximum CPU load for Tetris is 0.4. Moreover, the minimum CPU load of Gmail is 0 while that of Tetris is 0.05. The CPU usage for Gmail also has long low-value segments which look almost flat, indicating the period where there are no email composing/sending/receiving activities. However, such pattern is not observed in Tetris. Such differences might be exploited by an attacker who is capable of getting the resource usage patterns of apps to identify the type of app currently being used by a user. Thus, it is important to assess this privacy risk.

Inspired by this need, in this paper, we intend to answer the following research question: Can the resource (e.g., CPU and memory) usage pattern of mobile apps 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 investigate the resource usage patterns of apps and whether the difference in resource usage pattern is sufficient to classify different types of apps. For completeness, the resource usage patterns in both mobile device and cloud server are considered.

In particular, we propose two resource usage-based privacy attacks under the same framework that exploit the resource usage patterns of mobile apps to identify the type of apps based on machine learning techniques. To study the feasibility of the attacks, we first collect the resource usage patterns of various popular apps on a mobile device and test the attacks on the collected dataset. In mobile cloud computing, cloud servers know the resource usage of offloaded computations since they are the provider of the computing resources. To explore the potential privacy leakage from resource usage patterns of offloaded computations, we then collect resource usage patterns of various apps in a mobile cloud environment, and test the proposed attacks on the collected dataset. Note that 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 device and the cloud server.

The contribution of this paper is summarized as follows:

- To the best of our knowledge, this is the first work that explores whether the resource usage patterns of mobile apps can be used to infer the type of apps being used.
- This paper proposes two resource usage pattern-based privacy attacks using machine learning techniques. It investigates the feasibility of these attacks on datasets collected from both mobile devices and mobile cloud computing servers, and finds that the type of app can be inferred with high probability.

The remainder of this paper is organized as follows. Section II reviews related work. Section III describes our proposed attacks. Section IV and Section V describe the attack results on mobile devices and mobile cloud computing systems, respectively. Section VI concludes the paper.

II. Related Work

There has been much work in the traditional computer domain on predicting how much CPU time and how much memory a known program will use to better balance loads, e.g., [25]. Along that line, Shimizu et al. (2009) [26] predict the CPU time of a known program to better distribute the program in a distributed computing environment. However, these works predict the CPU usage of a known program. In contrast, our work infers the type of unknown mobile apps from their CPU and memory usage patterns in modern mobile devices and mobile cloud environments.

In the mobile security domain, despite the many works on mobile malware [5] and solutions [1], potential privacy risks with side channels such as resource usage patterns in mobile cloud computing scenarios have not received sufficient attention.

Xu et al. (2015) [7] profile 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 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 app signatures, they tested their implementation on real world traffic data and apps running on emulator. Yao et al. (2015) [8] develop a framework as an improvement to [7]. Dai et al. (2013) [23] also identify Android apps from network traces. They show 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 show that it is feasible to identify the app using the key/value found in the advertisement data flow. Moreover, their scheme is able to successfully predict when a type of app 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 thus inferring what users are doing, they only consider network traffic and do not consider CPU/memory usage which is the focus of our work. Also, they do not consider mobile cloud computing scenarios.

The growth of mobile cloud computing has been hindered by security and privacy concerns with it, and substantial work has been done in research organizations and academia to develop secure mobile cloud computing environments and infrastructures [2]. Ren et al. (2012) [4] discuss 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. Huang and Zhou (2011) [6] develop 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 provides security assurance, it does not address the potential privacy leakage to cloud server by exploiting resource usage patterns. Slamanig (2013) [24] propose 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 caused by exploiting resource usage patterns are among those unanswered concerns. This paper aims to investigate the possibility of privacy leakage resulting from resource usage patterns.

III. Resource Usage-Based Privacy Attacks

This section describes the proposed privacy attacks based on resource usage patterns.

A. Overview

The problem is that, given the resource usage time-series data of one mobile app, an 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. There are two reasons why the attacker wants 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 fan or even addicted to game, no matter which games he plays.

We 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 turns down to this: 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 paper, 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 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. 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. This 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. We choose some of the most widely used algorithms including k-nearest-neighbor, support vector machine, neural networks, classification tree, and Random Forests [11, 18]. In our experiments, Random Forests works best. Random Forests 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. Attack Exploration in Mobile Devices

This section explores the proposed attacks on dataset collected from a mobile device.

A. Dataset Collection

We 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. 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 can also be used 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 widely used by users. Then we choose the most popular four apps for each type of apps according to Google Play Store. It is reasonable to test widely used applications to reflect real life scenarios. A list of chosen applications is presented in Table II.

We run each app for 10 minutes and collect its resource usage data. When running an app, we perform 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 I.

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

To verify the correctness of collected data, a video of the mobile device screen during resource usage collection is recorded in the experiment. This video is used to confirm causal connections between the resource usage 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 validate the collected data.

TABLE I. Activities for Application Category

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 websites that the user usually visits, 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.

Then we plot the CPU load per second against the time. This plot of CPU usage patterns of the collected dataset shows that apps from the same type have similar patterns of CPU usage. In addition, the plot shows that apps from different

types have different patterns of CPU usage. The plot is not shown here due to the space limitation.

B. *Results of Attack Method 1*

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

TABLE II. LIST OF APPLICATIONS 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

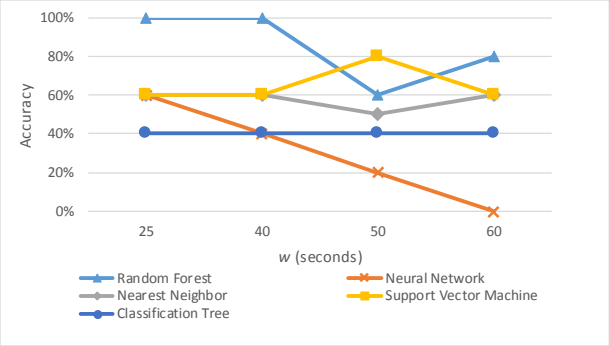

Fig. 2. The accuracy (y-axis) of Attack Method 1 using different learning algorithms and different *w* (x-axis)

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

App (true type)	Classification Results: Classified 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%	2/5 = 40%	3/5 = 60%	3/5 = 60%	2/5 = 40%

Figure 2 shows the results where different segment sizes (parameter *w*) are set. It can be seen that in most cases Random Forest outperforms the other four algorithms. Its accuracy achieves 100% when the segment size is 25 and 40. Table III shows more detailed results when the segment size *w* is set as 40. It can be seen that Random Forest's accuracy is evenly distributed across different types of apps. On the contrary, Support Vector Machine accuracy can differ significantly between categories (0% for Cloud Storage and 92.3% for Email Client).

C. *Results of Attack Method 2*

Then we tested Attack Method 2 that uses K-Nearest Neighbor with Dynamic Time Warping (DTW) distance since it is one of the machine learning algorithms that work very well on time series data. Parameter K is set as 3. The results are shown in Table IV. The accuracy with 3-Nearest-Neighbor+DTW is 80%. We also tested 3-Nearest-Neighbor+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 IV. The accuracy is the same as directly applying 3-Nearest-Neighbor+DTW to the entire data sample.

TABLE IV. 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)

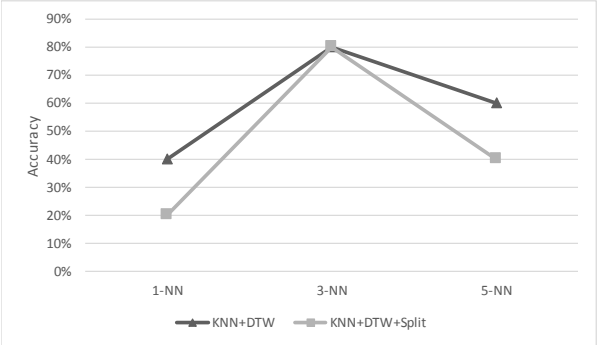

Fig. 3. Accuracy of k-Nearest Neighbors with Dynamic Time Warping Distance on Different k's on 10 Minutes CPU Usage Data

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

V. ATTACK EXPLORATION IN MOBILE CLOUD COMPUTING

This section explores the proposed attacks on dataset collected from a mobile cloud computing environment.

A. *Dataset Collection*

Many mobile cloud computing approaches have been proposed [19, 20, 21, 22] which can be divided into full offloading and partial offloading. In a full offloading method, the whole application computing will be offloaded to the cloud

server. This method does not require access to the application 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.

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 Section 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 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

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 4 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.

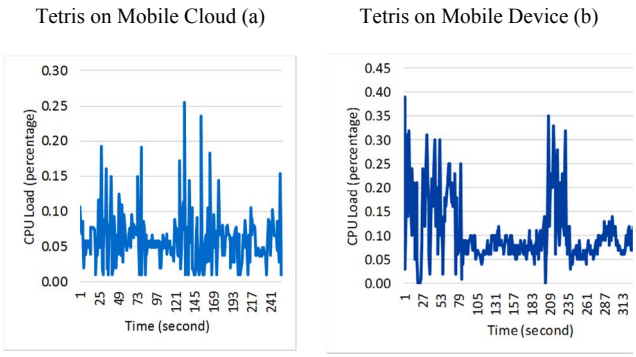

Fig. 4. CPU Usage Pattern of an app Tetris on Two Platforms

Similar to the mobile device setting, we plot the total CPU load per second against the time. This plot of CPU usage patterns shows that apps from the same type have similar patterns of CPU usage. In addition, it shows that apps from different types have different patterns of CPU usage. The plot is not shown here due to the space limitation.

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 (Attack Method 2) and Random Forest with 40-second data segments (Attack Method 1).

As can be observed in Table V and Table VI, Attack Method 2 performs better than Attack Method 1. Attack Method 2 can perform twice better than random guess (the accuracy of random guess is 33.3%). On the other hand, Attack Method 1 only performs as good as random guess.

TABLE V. 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)

TABLE VI. 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%

VI. CONCLUSION AND FUTURE WORK

This paper made an initial effort toward understanding the privacy leakage from the resource usage patterns of mobile apps. It 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 good accuracy. In the single mobile device environment, the accuracy reached 100%. In the mobile cloud computing environment, the accuracy reached 66.7%. This study discovered that privacy leakage by exploiting resource usage patterns is feasible.

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, based on data collected from many real-world users for a long time. Another direction is to evaluate the attacks for the partial offloading method of mobile cloud computing.

Besides assessing these attacks, another future direction is to develop countermeasures to mitigate these privacy leakages.

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 *Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI)*, 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*, 2011, pp. 301–314.

[4] K. Ren, C. Wang, and Q. Wang, "Security Challenges for the Public Cloud," *IEEE Internet Computing*, 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*, p. 3, 2011.

[6] D. Huang and Z. Zhou, "Secure data processing framework for mobile cloud computing," *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 *IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 1481–1489.

[8] H. Yao, "SAMPLES : Self Adaptive Mining of Persistent LExical Snippets for Classifying Mobile Application Traffic," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom)*, 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. USENIX conference on Operating Systems Design and Implementation (OSDI)*, 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 *ACM MobiSys*, 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.