Sensors and Materials, Vol. 35, No. 2 (2023) 579–588
MYU Tokyo

S & M 3196

A Behavior-based Scheme to Block Privacy Leakage on Smartphone Sensors When You Exercise

Liuqing Yang, Xiaorui Zhang, and Qiuju Wang*

Tourism College of Beijing Union University,
97# North Fourth Ring East Road, Chaoyang District, Beijing 100101, China

(Received October 15, 2022; accepted February 2, 2023)

Keywords: behavior-based access control, smartphone sensor, privacy leakage, block, API call monitoring

Smartphone sensors are widely used in the development of fitness, running, workout, or health applications (apps). However, smartphone sensors may increase the risk of privacy leakage when they bring convenience to users. Traditional access control mechanisms, such as Android Permission, cannot prevent authorized malicious apps from abusing sensor resources. In this paper, a novel behavior-based sensor access control scheme is presented. This scheme can further regulate the behavior of an app after it is authorized, so that it can only access sensor resources with secure behavior patterns (SBPs), and sensor-based privacy leakage may thereby be blocked. On the basis of user interface (UI) operation tracking and tagging, this scheme implements the dynamic perception of app sensor access behaviors. With a temporal logic known as temporal logic of causal knowledge (TLCK), we developed a method to construct the secure sensor access behavior pattern. Every sensor may be given a SBP. By comparing the dynamic sensor access behavior of an app with SBP, we can determine if the sensor access is secure. Moreover, by supervising the call stack of the app's sensor access application programming interface (API), we may timely block a sensor access when it is not secure. In this report, we also describe the implementation of a prototype defense system to analyze the effectiveness and efficiency of the scheme. The experimental results show that this scheme can effectively block the abnormal sensor access of an app with a performance overhead of about 10%.

1. Introduction

Health and exercise apps on smartphones are very popular. Almost all iPhones come with an iOS health app(1) pre-installed. There are hundreds of fitness, running, workout, health and other apps on Android and iOS app markets for users to download. These apps use the sensors carried by smartphones to deeply perceive user status information, conduct comprehensive interaction with users, and provide support for users' exercise.

However, the wide application of smartphone sensors also brings the risk of user privacy disclosure. If a sensor is abused, it may leak the user's private information while providing

*Corresponding author: e-mail: Wqiuju@126.com
https://doi.org/10.18494/SAM4173

support for the user's health and exercise. Therefore, the smartphone operating system provides permission management mechanisms, such as the Android permission mechanism, to enforce access control on sensors.(2) However, smartphones also carry sensors that are not controlled by access control mechanisms. These sensors may also be a channel for privacy disclosure. Ba *et al.* proposed a telephone eavesdropping attack based on an uncontrolled acceleration sensor.(3) Reddy *et al.* proposed an attack based on an acceleration sensor to monitor information on the state of movement of smartphone users.(4) Li *et al.* found an attempted theft based on location information from global positioning system (GPS) sensors.(5) Attackers may even break through the access control mechanisms of operating systems(6) to achieve sensor privacy theft. Elahi *et al.* revealed that attackers can use pre-installed system apps to steal private data, thus bypassing permission control.(7)

Therefore, designing and implementing more efficient and effective sensor privacy protection schemes is necessary. Bai *et al.* proposed a security control scheme for sensor access based on hooking a system's application programming interface (API) to mitigate the threat of privacy on disclosure.(8) Sliwa comprehensively analyzed the privacy and security issues of smartphone sensors.(9) Enck *et al.* introduced a TaintDroid that monitors the Android data flow using the stain tag tracking method to discover private data theft by malicious codes.(10)

In this paper, we introduce a behavior-based privacy protection scheme that may enforce a real-time and fine-grained access control on sensor resources to prevent theft of privacy information. Taking Microphone (MIC) as an example, in this paper, we describe the technical principle and implementation of this method. First, we define a secure behavior pattern (SBP) for MIC sensor access, which describes the behaviors that an app should follow in a MIC access with temporal logic of causal knowledge (TLCK).(11) Then, the dynamic behavior of an application is monitored, the results of which are compared with SBPs. Only matching access is allowed.

Compared with existing methods, this scheme possesses three advantages:

i. with respect to the diversity of malicious behavior, SBPs are more determined and may be expressed more easily, which decreases the complexity of the implementation of the defense scheme;
ii. it can stop the attacks before damage is caused, through real-time accessing control of the MIC;
iii. the scheme may work well in conjunction with existing protection mechanisms, such as Android Permission mechanism and TaintDroid.

2. Scheme Architecture

As shown in Fig. 1, the defense scheme contains three functional parts: App behavior monitoring, SBP definition, and access control implementation in sensor access interfaces. As with permission mechanisms, every sensor resource is assigned an SBP. Owing to limited space, in this paper, only the SBP for MIC is provided. An SBP is a temporal logic of API calls and system event handling process. The SBP is defined according to access features of the specific sensor. When the system is started, all SBPs are loaded into the system memory space.

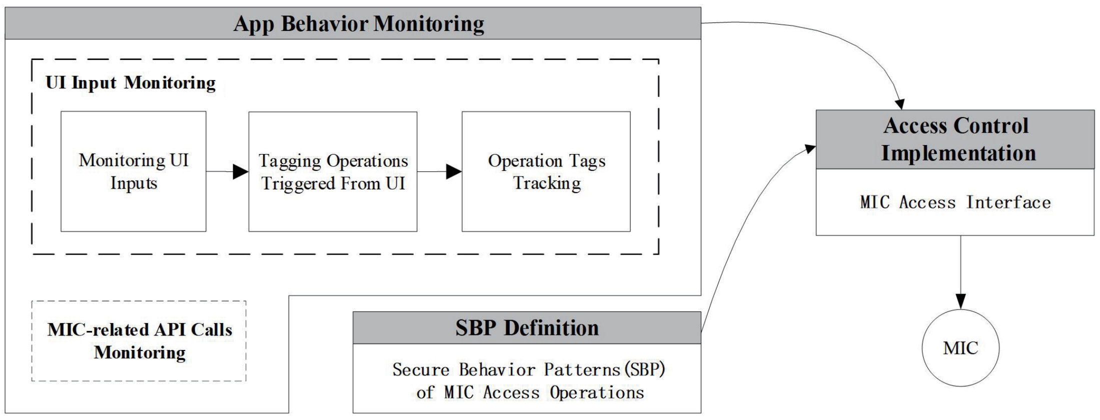
App Behavior Monitoring
UI Input Monitoring
Monitoring UI Inputs
Tagging Operations Triggered From UI
Operation Tags Tracking
MIC-related API Calls Monitoring
SBP Definition
Secure Behavior Patterns (SBP) of MIC Access Operations
Access Control Implementation
MIC Access Interface
MIC

Fig. 1. Architecture of the defense scheme.

App behavior monitoring is in charge of acquiring the dynamic app behavior information. As described, the behavior of an app includes API calls and system event handling. We realize app dynamic behavior acquisition through MIC-related API call monitoring. As for MIC abuse detection, the system event handling is primarily user interface (UI) event handling. All the app codes are interpreted and executed by the Android runtime (ART) virtual machine (VM).(12) Through ART extension, the complete dynamic behavior information of an app may be obtained. ART instance is a thread inside a specific app process. Consequently, monitoring operations may be limited only to the specific process, such as the process associated with MIC access permission. Thus, the comprehensive performance of the system may not be seriously impacted. When the behavior of an app is observed, all the SBPs are scanned one by one. If a SBP is matched, permission to access the corresponding sensor is granted to this app. To avoid replay attacks, the sensor access must be completed within a predetermined time. Otherwise, the capability expires.

When an app initiates sensor access, besides the permission check, the access capability of its process is also checked. Even if the app has been assigned a corresponding permission, the sensor access is prohibited in the case in which the capability has not been granted to the app process. Different from permission mechanisms, in this scheme, the access capability is assigned dynamically based on access behavior checking. Therefore, the scheme can enforce real-time and fine-grained access control.

3. Prototype System Implementation

In this section, we describe the key mechanisms of implementation of the prototype system for this scheme.

3.1 SBP construction of MIC operations

A common feature in all sensor-based voice privacy theft attacks is that the attacks are all carried out through initiating voice capturing automatically in the attack code without the awareness of the user. If we provide access control to the MIC sensor and deny all automatically initiated voice capturing operations, these attacks can be held back successfully. The SBPs of MIC operations are defined according to this finding. In this section, we describe how to specify SBPs in terms of system events interposed by temporal and logical operators. The specification of SBPs is the first step in our behavior-based defense system.

3.1.1 Overview of TLCK

TLCK is a temporal logic that allows the description of propositions whose evaluation depends on time, making it suitable for describing sequences of events and properties of correlated behaviors. We formally define a behavior signature as a finite set of propositional variables interposed with TLCK, where each variable (when true) confirms either the calling of a single or an aggregation of Java API, or the attribute of the methods being called or the threads calling the methods.

The logical operators 'or ($\vee$)' and 'and ($\wedge$)' are defined as usual. The temporal operators defined using past-time logic are as follows:

- $\boxtimes_t$: true at time t.
- $\triangle_t$: true at some instant before t.
- $\square_{t-k}^{t}$:true at some instant in the interval $[t - k, t]$.

Assuming that we have variables X, Y, and Z, then we can define a temporal logic with TLCK as follows:

$$\boxtimes_t (X) \vee [\triangle_t (Y) \wedge \square_{t-k}^{t} (Z)\,]. \tag{1}$$

The meaning of Eq. (1) is that the event X happens at time t, or the event Y occurs before t, and the event Z is true in the time interval $t - k$ and t. Through extracting the behavioral characteristics of MIC operations, we can define the behavior variables and temporal logic. We can then define the SBPs of MIC operations.

3.1.2 SBP construction for MIC operations

In this section, we define the SBPs of MIC operations with TLCK. We first identify a set of atomic propositional variables.

- **Key Pressed:** A key pressing operation is initiated by the user.
- **Screen Touched:** A screen touching operation is initiated by the user.
- **UIResponse:** An event handle function of an UI operation is triggered, such as onTouchEvent(), onKeyDown(), onKeyUp().
- **VoiceRecord:** A Voice capturing API is called, such as MediaRecorde.start(), or VoiceRecord.startRecording().
- **MethodTainted(M):** The method M has an Mtag tag set to 1, which means that the method calling is triggered by the UI event handle function. Mtag is a tag that we add to the method call stack.

- **ThreadTainted(T):** The thread T has Ttag set to 1, which means that the execution of the thread is triggered by the UI event handle function. Ttag is a tag that we add to the thread call stack.

The SBPs of MIC operations are defined as follows:

$$\begin{gathered}\boxtimes_t (\text{KeyPressed} \vee \text{ScreenTouched}) \wedge \Box_t^{t+5} (\text{UIResponse}) \\ \wedge \Box_t^{t+k} \{\text{VoiceRecord} \wedge [\text{MethodTainted}(\text{VoiceRecordAPI}) \\ \vee \text{ThreadTainted}(\text{ThreadCallVoiceRecordAPI})]\}.\end{gathered} \tag{2}$$

Equation (2) defines a secure control flow of a secure MIC access. First, the user presses a button or presses the touch screen at time *t*. Then, a UI input handle function is triggered 5 ms after the user's UI operation. Finally, a voice capturing API is called, and the Mtag tag of the API method is set to 1 or the thread calling the API has Ttag tag set to 1.

Many experiments demonstrated that the time interval between either two UI operations is greater than 15 ms, while the time interval between a UI operation and a handle function being triggered is less than 5 ms. Therefore, the first two temporal logics in Eq. (2) can be used to determine whether the event handle function is triggered by a real UI operation. The third temporal logic can be used to determine whether a voice capturing operation is launched and whether it is triggered through a UI event handle function. Combining these logics, we can determine whether a voice capturing operation is initiated by the user manually or launched by the attack code automatically.

3.2 Real-time monitoring and tracking the UI operation of the app

Real-time monitoring of the behavior of apps is also an important part of the defense system. The monitoring results were compared with the SBPs defined in Sect. 3.1. All the behaviors such as the UI operations and the voice capturing API calls in the SBPs should be monitored.

In this section, we introduce a UI input tagging and tracking technology, which is used to determine whether an API calling is triggered by the user manually through the UI. We first detect whether a real UI operation is occurring by monitoring the Linux input subsystem. We then use time information to detect whether an event handle function was triggered by the UI operation. Finally, we introduce one tagging and tracking technology to track the API callings triggered by the event handle function.

In Android, the UI operations, such as Keyboard and Touchscreen, are processed based on the Linux input subsystem. The UI input event is first written into '/dev/input/' by a Linux kernel when a Keyboard or Touchscreen operation occurs. Then the 'SystemServer' process in Android captures the event by reading '/dev/input/' and broadcasts the event. Finally, the event handle functions in an Android app, such as 'onTouchEvent', 'onKeyDown', 'onKeyUp', are triggered and the corresponding operations are taken.

Detection of the UI operations is carried out according to the processing flow of the Linux input subsystem, which includes the following two steps:

i. Monitoring the writing operations to '/dev/input/' in the Linux kernel. Based on the hypothesis that the kernel is secure, this operation cannot be forged by the attackers. Therefore, if a writing operation is histed, a real UI operation occurs.
ii. Detecting the event handle function triggered by the UI operation. The processing flow from the input event recorded by the Linux kernel to the handle function triggered involves multiple processes, and thoroughly tracking this flow is time-consuming. Through hundreds of experiments, we found that the time span between the above two operations is less than 5 ms, while the time span between either of two UI operations is more than 15 ms. Therefore, it is possible and convenient to use the time information in the event to determine whether an event handle function is triggered by the specified input action. This is the approach employed in our defense system.

We introduce a three-level tagging and tracking technology to detect whether an API calling is triggered in the event handle function. In an Android system, there are three ways to initiate an API calling in the event handle function: i. directly calling the API in the handle function; ii. creating a new thread to launch API calling; and iii. sending a message to tell an existing thread to call the API. As shown in Fig. 2, we introduce three tags to mark the above three scenarios: 'Mtag', 'Ttag', and 'Msgtag'. We add an 'Mtag' tag to each method in the interpreted stack to indicate whether it is triggered by an event handle function. The 'Mtag' of a method is set to 1 if the method is an event handle function, or to 'Mtag' of the calling method is 1. In the interpreted stack, all methods in the position between pushing and popping of one method are called by the method.

We add a 'Ttag' to each thread; threads created in one method have the 'Ttag' set to 1 if the 'Mtag' of the method is 1; otherwise, the 'Ttag' will be set to 0. We add an 'Msgtag' to the message, and all messages sent in the method with 'Mtag' 1 or in the thread with 'Ttag' 1 have the 'Msgtag' set to 1. Then we check the message's 'Msgtag' when the message handle function is called and set the function's 'Mtag' to 1 if the message's 'Msgtag' tag is 1. With this tagging approach, we can track all operations triggered by an event handle function, either with 'Mtag' set to 1, or with 'Ttag' set to 1. In contrast, if an action, including the voice capturing action, is

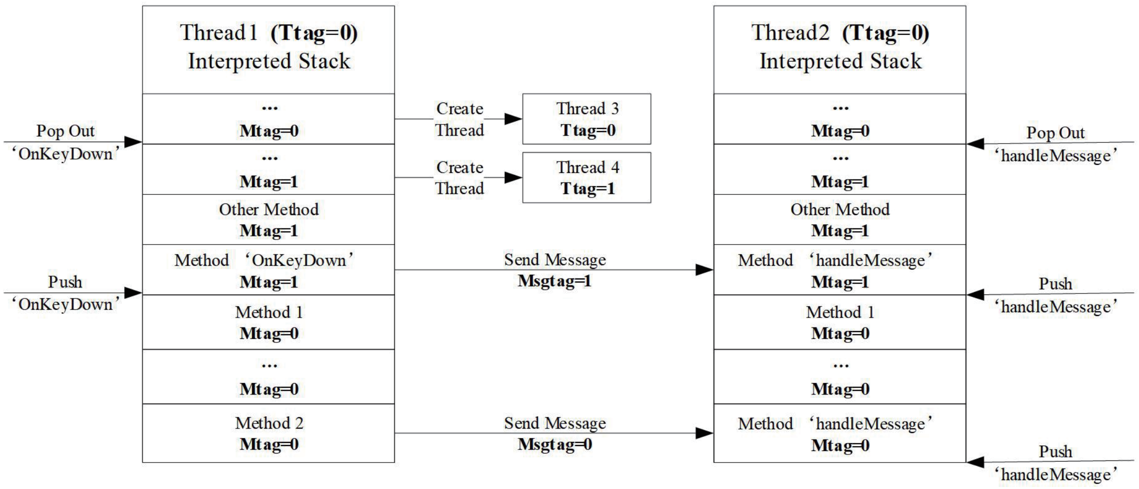

Fig. 2. Tagging and tracking the operations triggered in the event handle functions.

launched stealthily, both the ‘Mtag’ of the method and the ‘Ttag’ of the thread calling the method will have a value of 0. With these technologies, we can determine whether an API calling is initiated by the user manually or by the attack code automatically, making the assumption that the kernel is secure.

3.3 Monitoring MIC sensor access API calls

Android apps written with Java codes are first compiled into ART bytecodes and then executed in the ART VM. Thus, all Java methods called in an app can be monitored in the ART VM. In the defense system, we add monitoring codes to the ART VM to detect if a voice access API calling has been launched.

As shown in Fig. 3, there are two ways to call a Java API in an Android: by calling directly from Java codes, and by calling from native codes via JNI. In both ways, the methods are pushed into the interpreted stack before BEING executed. Thus, the pushing operation of the interpreted stack is a suitable point to monitor the Java API calls in an app. In fact, the pushing operation of the interpreted stack is implemented at two points. For the API calls in Java codes, the pushing-operation codes are included in the hardware-related part of the ART VM. As for the API calls from native codes, the pushing operation is complete in the common part of the ART VM. Thus, the monitoring codes are added to these two points to completely cover all pushing operations on the interpreted stack. Through parsing an item in the interpreted stack, the parameters of an API call can be monitored effectively. The position of the item in the interpreted stack reflects the sequence of API calls.

4. Scheme Validation and Evaluation

In this section, we report the experiments to evaluate the effectiveness and efficiency of the defense prototype system. Since we need to modify the kernel code to implement our defense prototype system, we chose the Google smartphone with open-source kernel to deploy our defense prototype system. Our experimental platform is the Google Pixel smartphone with the Android 12 operating system.

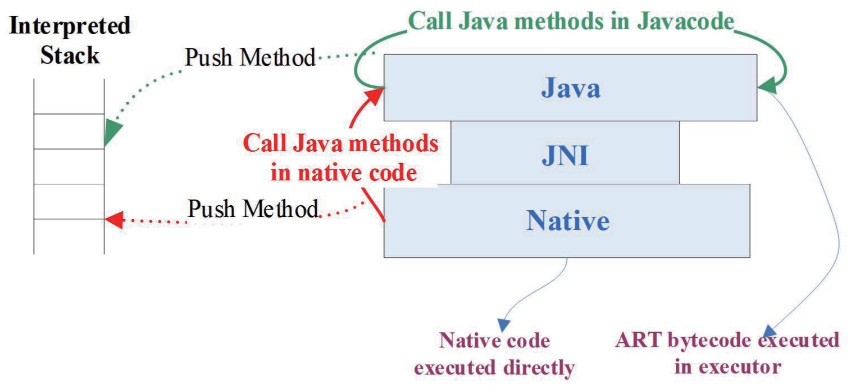
Interpreted Stack
Call Java methods in Javacode
Push Method
Java
JNI
Call Java methods in native code
Push Method
Native
Native code executed directly
ART bytecode executed in executor

Fig. 3. (Color online) API call stack in the ART VM of an Android.

4.1 Effectiveness

To test the effectiveness of the defense prototype system, we introduced a typical audio information attack case named Cyber-Physical Voice Theft (CPVT) presented in Ref. 13. It has been reported that CPVT may bypass the popular mobile antivirus software.(13) We deployed Kaspersky,(14) 360 Mobilesafe,(15) AVG,(16) and our defense prototype system on the Google experimental smartphone and observed the defense effect of different antivirus software against the CPVT attacks. The experimental results are shown in Table 1. The behavior-based malicious app detect method implemented in our defense prototype system effectively found and stopped the abuse of MIC sensors.

4.2 Efficiency

The defense system was implemented as an extension of the ART VM. Therefore, we evaluated the efficiency by comparing it with a normal Android ART VM. ART VM is also a type of Java VM. CaffeineMark 3.0(17) for Android is adopted to generate the scores of Java Microbenchmark, which is a classic metric used to evaluate the performance of Java VM. CaffeineMark3.0 uses an internal scoring metric only useful for relative comparisons. The experimental results are shown in Fig. 4.

We first evaluated the total performance overhead of the defense system, and then evaluated the respective performance overhead incurred by the two major function modules of the defense

Table 1
Effectiveness comparison with mainstream antivirus software.

Antivirus Software	Defense Effect	
	Voice capturing	Voice data sending
Kaspersky	Yes	Yes
360 Mobilesafe	Yes	Yes
AVG	Yes	Yes
Our defense system	No	No

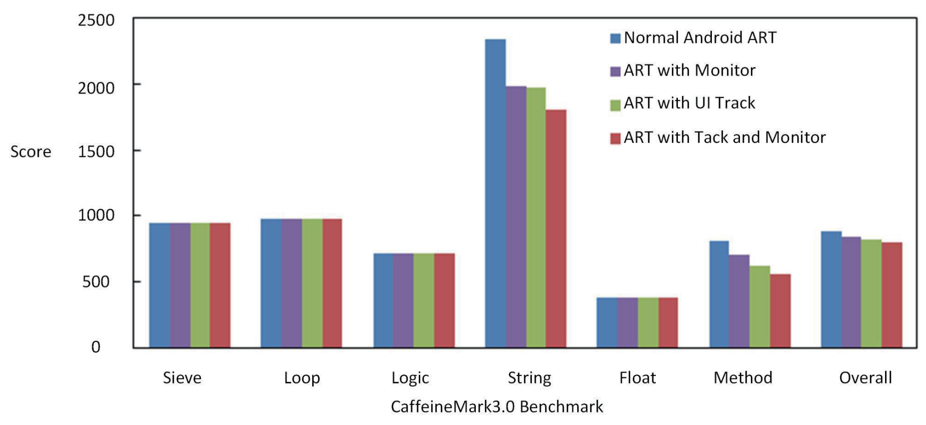

Fig. 4. (Color online) Overhead on a normal Android ART VM and an ART VM with a defense system.

system as discussed in Sects. 3.2 and 3.1, namely, UI operation tracking and API call monitoring. As illustrated in Fig. 4, the experimental results on the overhead of the two functions are shown as ‘ART VM with UI Tracking’ and ‘ART VM with API monitoring’, and the results for the whole system are shown as ‘ART VM with UI Tracking and API Monitoring’. The experimental results for the performance of the original ART VM for comparison are shown as ‘Normal Android ART VM’. The results are consistent with design expectations. The overhead incurred is almost zero for the benchmarks dominated by arithmetic and logic operations; the string and method benchmark experience more overhead, which is caused by the memory comparisons that occur for the method names and the monitoring of API calls.

The ‘overall’ benchmark indicates the cumulative score across all individual benchmarks. CaffeineMark3.0 documentation indicates that the score of ‘overall’ benchmark roughly corresponds to the number of Java instructions executed per second. Here, the normal Android system has an average score of 880, while the score of the entire defense system is 790. The scores of the ART VM with API call monitoring and UI tracking are 835 and 818, respectively. The total overhead for the defense system is almost 10.23%, and the overhead caused by UI tracking is more than that caused by API monitoring.

5. Conclusions

We presented a behavior-based sensor access control mechanism to prevent the privacy disclosure caused by a sensor. Taking advantage of the tracking and tag technologies of UI operations, we propose a dynamic acquisition method for app sensor access behavior patterns. By this method, we may dynamically observe the behavior pattern of sensor access by an app. On the basis of the TLCK model, taking MIC sensors as an example, we proposed a model to describe the pattern of secure access to smartphone sensors. With this model, we defined the SBP for every sensor. In sensor access, we compared the dynamic real-time sensor access behavior with SBP to block insecure sensor access. Compared with traditional access control mechanisms, the behavior-based mechanisms can regulate sensor access more easily and in a fine-grained fashion. Even if the app has the corresponding sensor access permissions, it must follow the SBP to access sensor resources. In this report, we also presented a behavior-based sensor access control enforcement mechanism through monitoring sensor API calls. This method parses the API call stack, monitors the sensor access API, and introduces a behavior-based access control mechanism to enforce access control. The experimental results on the prototype defense system show that the behavior-based access control mechanism proposed can effectively prevent the abuse of sensor resources and block user privacy disclosure with reasonable overhead.

Acknowledgments

The authors thank the anonymous reviewers and editors for their valuable comments and suggestions. This work was supported by the Science and Technology Innovation Service Capacity Building Project of the Beijing Municipal Education Commission (Grant No. KM201811417012).

References

1 See More of Yourself in Health: https://www.apple.com/ios/health/ (accessed October 2022).
2 Permissions on Android: https://developer.android.com/guide/topics/permissions/overview (accessed October 2022).
3 Z. Ba, T. Zheng, Z. Qin, H. Yu, L. Liu, B. Li, X. Liu, and K. Ren: Proc. 26th Annu. Int. Conf. Mobile Computing and Networking (MobiCom 2020) 1–2.
4 S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava: ACM Trans. Sens. Networks **6** (2010) 1. https://doi.org/10.1145/1689239.1689243
5 Z. Li, Q. Pei, I. Markwood, Y. Liu, M. Pan, and H. Li: IEEE Trans. Veh. Technol. **67** (2018) 5042. https://doi.org/10.1109/TVT.2018.2800123
6 K. Han, Y. Lee, B. Jiang, and E. Im: Int. J. E-Entrepreneurship Innov. **4** (2013) 16. https://doi.org/10.4018/jeei.2013010102
7 H. Elahi, G. Wang, and X. Li: Proc. Int. Conf. Security, Privacy and Anonymity in Computation, Communication and Storage (SpaCCS, 2017) 169–185.
8 X. Bai, J. Yin, and Y. P. Wang: EURASIP J. Inf. Secur. **10** (2017) 1. https://doi.org/10.1186/s13635-017-0061-8
9 J. Sliwa: Intelligent Data Sensing and Processing for Health and Well-being Applications, M. Wister, P. Pancardo, J. A.Hernández, Eds. (Academic Press, New York, 2018) Chap. 7.
10 W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth: ACM Trans. Comput. Syst. **32** (2014) 99. https://doi.org/10.1145/2619091
11 W. Penczek: Logic J. IGPL **8** (2000) 87. https://doi.org/10.1.1.37.2389
12 A. M. M. Soares and R. T. S Junior: Proc. Int. Conf. Information Systems Security and Privacy (ICISSP, 2017) 130–147.
13 L. Lei, Y. Wang, J. Zhou, L. Wang, and Z. Zhang: Proc. 12th IEEE Int. Conf. Trust, Security and Privacy in Computing and Communications (TrustCOM, 2013) 126–133.
14 Kaspersky Cyber Security Solutions for Home & Business: https://www.kaspersky.com (accessed October 2022).
15 360 Mobilesafe: https://shouji.360.cn/v6/index.html (accessed October 2022).
16 Free Antivirus for Android | AVG Mobile Security App: https://www.avg.com/en-us/antivirus-for-android (accessed October 2022).
17 CaffeineMark 3.0 Information: http://www.benchmarkhq.ru/cm30/info.html (accessed October 2022).

About the Authors

Liuqing Yang received her B.S. degree from Huazhong University of Science and Technology, China, in 2001, her M.S. degree from Central University of Finance and Economics, China, in 2005, and her Ph.D. degree from Beijing University of Technology, China, in 2014. Since 2014, she has been an assistant professor at Tourism College of Beijing Union University, China. Her research interests are in tourism and exhibition economy management and tourism and exhibition informatization. (liuqingyang@buu.edu.cn)

Xiaorui Zhang received his B.S. degree from Shanxi University of Finance and Economics University, China, in 2022. Since 2022 he has been a postgraduate at Beijing Union University, China. His research interests are in industrial heritage tourism, industrial tourism, and place meaning. (20221120210522@buu.edu.cn)

Qiuju Wang received her Ph.D. degree from Northeast Agricultural University, China, in 2007. She is currently a professor in the School of Tourism of Beijing Union University. Her research interests are in industrial heritage tourism and tourism economy. (wqiuju@126.com)