Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 8090652, 18 pages
https://doi.org/10.1155/2022/8090652

Research Article

Behavicker: Eavesdropping Computer-Usage Activities through Acoustic Side Channel

Mengqi Chen, Jiawei Lin, WeiFeng Liu, and Kaishun Wu

College of Computer Science and Software Engineering, Shenzhen University, China

Correspondence should be addressed to Mengqi Chen; chenmengqi2017@email.szu.edu.cn

Received 17 June 2022; Accepted 18 August 2022; Published 21 September 2022

Academic Editor: Pengfei Wang

Computers are widely used for business and entertainment purposes throughout our modern lives. Computer kits provide a variety of services such as text processing, programming, shopping, and gaming. Computers have greatly enhanced the quality of our lives; however, we discover an often-overlooked fact that engaging in computer-related activities may be eavesdropped upon by an attacker by sniffing the emitted acoustic signals from keyboard and mouse. The activity of eavesdropping via acoustic side channel has lower requirements in terms of hardware instrumentation and is easier to implement in real-world applications than other side channel attacks that have been presented in previous work. In this paper, we design and implement a system, namely, Behavicker, to validate the feasibility of this kind of attack. Unlike conventional activity recognition, Behavicker infers high-level computer-usage activities with a semantics-preserving multiscale learning scheme, based on the recognition of basic keyboard and mouse events including *left click*, *right click*, *middle click*, *scrolling up*, and *scrolling down*. Real-world experiments show that Behavicker can recognize six *interaction events* with an accuracy of 88.3% and infer computer-usage activities with an accuracy of 82.7% in an indoor environment.

1. Introduction

It is still the case that desktop computers dominate office environments despite the rapid penetration of mobile devices. A recent survey [1] shows that desktop PCs remain the most commonly used office facility for online shopping, communicating, document editing, etc. Consequently, desktops contain increasing personal information of users, of which even computer-usage activities reflect users' privacy. Traditional privacy eavesdropping methods require installing malicious software on target desktops which can be easily detected and blocked. However, researchers have revealed some more imperceptible eavesdropping attacks in recent days which steal private information during using desktops with various sensors such as microphone [2] and magnetic sensor [3]. Most of them focus on recognizing keystrokes and infer the input contents of an attacker [2–4]. Different from them, we investigate another unexplored eavesdropping method which infers computer-usage activities on a desktop from the audio recorded by a smartphone placed nearby. Specifically, we consider such a practical attack scenario, in which a hacked smartphone is placed near a desktop and continuously records acoustic signals induced by a user's *interactions* with the desktop via a keyboard and mouse. With the acoustic signals transmitted to and analyzed on a sever, the attacker is able to know what programs and services the victim is using in real time.

Someone may wonder what benefits can an attacker obtain from such a kind of eavesdropping, especially considering that previous works can infer keystrokes [2–4]. We admit that keystroke recognition can obtain more fine-grained information compared with our high-level computer-usage activity eavesdropping. Nevertheless, it is rather difficult to identify valuable information from a large amount of keystroke sequences without knowing what specific programs or services the target is using. For example, when an attacker wants to eavesdrop the user name and password of a target's bank account, it is labour-intensive to extract corresponding information hidden in long sequences. Therefore, computer-usage recognition makes the attack scheme more threatening. Such high-level recognition can be the prior process of the attack workflow. Filtering out unwanted information reduces the difficulty and increases the efficiency of eavesdropping. Further, not only did we consider keystrokes but also mouse clicks.

Based on the above, this paper proposes Behavicker, an acoustic side channel eavesdropping system that monitors common computer-usage activities such as texting, gaming, and web browsing. Different from conventional activity recognition methods, we decompose an high-level computer-usage activity into six low-level *meta events* including *keystroke*, *left click*, *right click*, *middle click*, *scrolling up*, and *scrolling down*. Our insights are twofold. For one thing, all the human-computer interaction activities are composed of part of the above events but differ from each other in the distribution patterns. For example, text processing involves the use of a substantial number of keys, whereas web browsing is likely to rely primarily on mouse clicks. It is feasible to distinguish different computer-usage activities based on event distributions. For another thing, the mechanical design of keyboards and mice results in distinct acoustic properties associated with interaction events, which makes it possible to recognize *meta events*.

However, in order to realize such an idea, we need to deal with the following two key challenges. First, as it is nearly impossible to get labeled training data from a target, it is challenging to design an accurate classification model. To handle this, we propose an iterative model adaptation method in the learning process which feedbacks samples with high confidence output by an initial inaccurate classification model, in order to boost the model gradually. Second, considering the diversity of desktop applications and hardware, it is difficult to devise a feasible framework for computer-usage activity recognition. We design a hierarchical learning scheme to distinguish different computer-usage activities. In the first layer, we classify the acoustic recordings into six basic interaction events. In the second layer, we design a tree-structured classifier to identify different computer-usage activities from basic interaction events via time-series analysis.

We implement Behavicker with different keyboard and mouse pairs and conduct comprehensive evaluation in various settings. Our experiments reveal that Behavicker can recognize the aforementioned six *meta events* with an accuracy of 88.3% without any training samples from a target at the beginning. More importantly, Behavicker can identify seven different categories of computer-usage activities including *instant messaging*, *programming*, *text processing*, *gaming*, *reading*, *online shopping*, and *designing* with an accuracy of 82.7%, by analyzing the temporal distribution of *meta events*. In a summary, our contribution includes the following:

(i) An acoustic side channel is first used to demonstrate the eavesdropping of desktop computer-related activity. Specifically, we have designed acoustic processing schemes, keyboard, and mouse interaction-event recognition schemes and computer-usage activity recognition schemes appropriate for eavesdropping

(ii) To evaluate the effectiveness of our proposed eavesdropping scheme, we implement Behavicker on commodity devices and conduct extensive real-world experiments with 20 participants on 5 pairs of keyboards and mice in 3 indoor environments. The experimental results have shown that it is possible to recognize up to 88.3% of 6 user interaction events in real-world office environments and 82.7% of seven computer-usage tasks

In the rest of this paper, we review related work in Related Work, present the scope and overview of Behavicker in Overview, and introduce its detailed design in Acoustic-Based Interaction Event Recognition and Computer Usage Recognition. Evaluation presents the evaluations, and finally, we conclude this work in Conclusion.

2. Related Work

2.1. Computer Eavesdropping via Side Channels. There are many potential side channels to eavesdrop computer contents or usage, including electromagnetic [5, 6], optical [7, 8], vibrational [9, 10], and acoustic [2, 4, 11–13]. Eavesdropping via electromagnetic and optical emanations needs specialized equipment [14]. In contrast, vibrational and acoustic side channels are viable on commodity smart devices [2, 4, 9–13]. For instance, Marquardt et al. [10] and de Souza Faria et al. [9] make use of commercial accelerometers to sense vibrations of pressing keys and decode text input via a nearby keyboard or ATM keypad. Asonov et al. [11] observe that the sound of keystrokes differs from key to key and build a supervised learning model to recognize keystrokes. A series of following studies are then proposed to snoop keyboard input via acoustic side channels, by combining different techniques such as time difference of arrival (TDOA) [2–4], the language model [13], and a dictionary model [12]. Our work is inspired by previous efforts on acoustic side channels. In comparison, rather than focusing on locating which keystroke a user is typing, this work is aimed at inferring high-level semantic computer-usage activities from keystroke, mouse clicking, and mouse scrolling patterns. It makes previous work easier for an attacker to further extract wanted information from target activities compared to a large amount of unlabeled keystroke sequences.

2.2. Acoustic Sensing with Smartphones. The built-in microphone of smartphones has been used to perceive more than human speech. Researchers have exploited it to sense various nonspeech activities including but not limited to food intake [15], tooth brush [16], sleep [17], and breath-related symptoms [18]. For instance, Hao et al. [19] and Gu et al. [17] use built-in microphone of a smartphone to detect sounds caused by activities during sleep and assess the quality of sleep. Ren et al. [20] achieve fine-grained sleep monitoring by detecting breathing rates and sleeping events with smartphone earphones. Lu et al. [21] propose a scalable framework for classifying sound events on smartphones. Similarly, sound can be used to analyze the performance of tooth brush [16] and food intake [15]. SymDetector [18] detects sound-related respiratory symptoms occurred in a user's daily life, including sneeze, cough, sniffle, and throat clearing. StressSense [22] recognizes human stress unobtrusively from voice using smartphones. [23] makes use of Doppler effect to sense gestures and achieve interaction between devices. Our work also tries to recognize nonspeech activities via acoustic sensing. In particular, we design proper audio processing pipelines to detect, segment, and identify 6 user interaction events with keyboards and mice for computer-usage activity tracking.

3. Overview

We aim to recognize computer-usage activities in an office environment by analyzing patterns of user interface events such as keystrokes and mouse clicks, which can be perceived by microphones via acoustic sensing. This work serves as a feasibility study on the potential privacy leakage of computer usage (e.g., either the user is working or gaming) caused by an acoustic side channel (e.g., the microphone of a smartphone placed near the computer in use).

3.1. Computer-Usage Activities. Since exhaustive coverage of all computer-usage activities is impossible, we restrict our scope to 7 categories of computer usage covering 20 computer software (In the rest of this paper, we regard activities with different software to be different activities, even though they belong to the same category.) as a feasibility study. The 18 software are selected based on a questionnaire survey from 150 college students and staffs (56 females and 94 males, aged 20 to 47) in our department. The participants were asked to list 30 most frequently used computer software in their daily life. We list 18 most frequently mentioned software and classify them into 7 computer-usage activities below. The numbers in the bracket represent the number of participants who list the software as “commonly used in daily life” in the survey.

(i) Instant messaging: WeChat (M_1, 150); QQ (M_2, 120); Jingdong Chat (M_3, 150); and Taobao Chat (M_4, 130)

(ii) Programming: Java programming with Eclipse (P_1, 123) and Python programming with PyCharm (P_2, 118)

(iii) Text processing: Microsoft default notepad (T_1, 106) and Microsoft Word (T_2, 150)

(iv) Gaming: Tetris (G_{a1}, 98); Dino Runner (G_{a2}, 87); Landlord (G_{a3}, 102); and Plants vs. Zombies (G_{a4}, 110)

(v) Reading: News on Baidu (R_1, 88); Novels on Qidian (R_2, 90); and Comics on Netease (R_3, 102)

(vi) Online shopping: Jingdong (S_1, 150) and Taobao (S_2, 145)

(vii) Designing: Photoshop (D_1, 81)

We make two notes on the survey results and the categories of computer-usage activities. (i) We focus on users of the Windows operating system because it dominates the desktop operating system in market [24]. (ii) Jingdong and Taobao listed in the online shopping category are popular online shopping platforms in China. They both provide instant messaging functions, i.e., Jingdong Chat and Taobao Chat. In this work, we separate the instant messaging activities during online shopping and other online shopping activities such as browsing products.

3.2. User Interaction Events. We focus on 6 common user *interaction events* with the keyboard and the mouse, including keystrokes (KS), clicking left mouse button (i.e., *left click*, CLM), clicking right mouse button (i.e., *right click*, CRM), scrolling mouse wheel up (i.e., *scrolling up*, SWU), scrolling mouse wheel down (i.e., *scrolling down*, SWD), and clicking mouse wheel (i.e., *middle click*, CMW). Behavicker tries to distinguish these 6 events for two reasons.

(i) Since desktop users mainly use the keyboard and the mouse to interact with the computer, most computer-usage activities involve some of the user *interaction events* above. Furthermore, it is possible to distinguish different computer-usage activities from the temporal patterns of these *interaction events* because different categories of computer-usage activities differ in the interaction logics

(ii) Due to the differences in material and mechanics, these *interaction events* tend to generate sounds that exhibit distinctive characteristics. Thus, it is possible to differentiate these events via acoustic sensing

3.3. Behavicker Overview. Behavicker consists of two functional modules, *acoustic-based interaction event recognition* and *computer-usage recognition*. At a high level, Behavicker collects acoustic signals via microphones and detects and recognizes the 6 *interaction events*. Sequences of user *interaction events* are then analyzed to identify different categories of computer-usage activities.

Figure 1 shows the overall data processing architecture of Behavicker. The *acoustic-based interaction event recognition* module applies single processing and machine learning techniques catered for keyboard and mouse interaction-induced sounds. The raw acoustic streams are first preprocessed to remove noises and improve the signal-to-noise ratio (SNR). Then, an adaptive threshold scheme is adopted to segment acoustic events of interests. Finally, a tree-structured classifier is trained to identify different keyboard and mouse *interaction events*. Since the labeled data in an eavesdropping setting are usually limited, a model adaptation scheme is proposed to boost the training of the classifier with limited labeled data. The *computer-usage recognition* module takes the streams of keyboard and mouse *interaction events* as input and applies a hierarchical classifiers to distinguish different computer-usage activities via time-series analysis. As next, we elaborate on the details of each module in sequel.

4. Acoustic-Based Interaction Event Recognition

This section explains how Behavicker processes raw acoustic signals to detect and identify common keyboard and mouse interaction events.

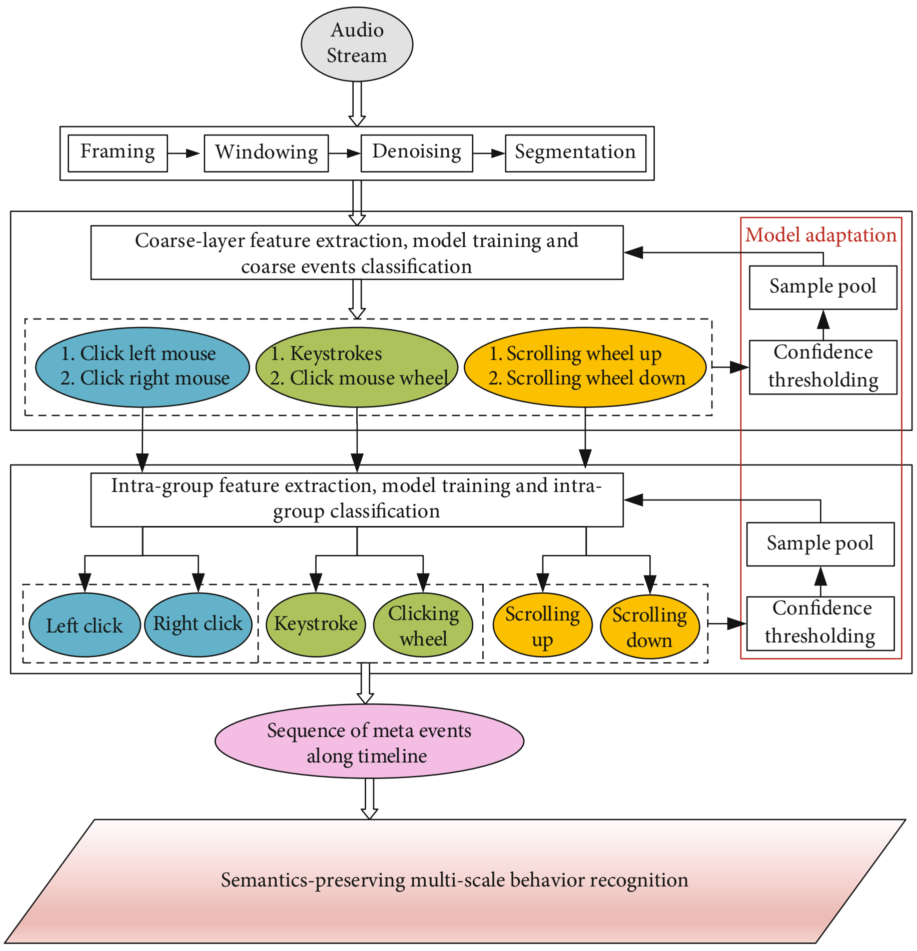

Audio Stream
Framing
Windowing
Denoising
Segmentation
Coarse-layer feature extraction, model training and coarse events classification
Model adaptation
Sample pool
Confidence thresholding
1. Click left mouse 2. Click right mouse
1. Keystrokes 2. Click mouse wheel
1. Scrolling wheel up 2. Scrolling wheel down
Intra-group feature extraction, model training and intra-group classification
Sample pool
Confidence thresholding
Left click
Right click
Keystroke
Clicking wheel
Scrolling up
Scrolling down
Sequence of meta events along timeline
Semantics-preserving multi-scale behavior recognition

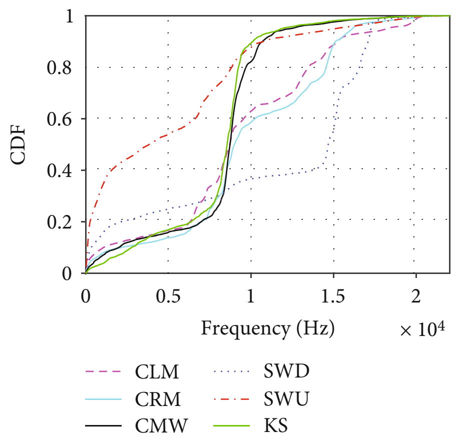
FIGURE 1: The system architecture of Behavicker.

4.1. Interaction Event Detection

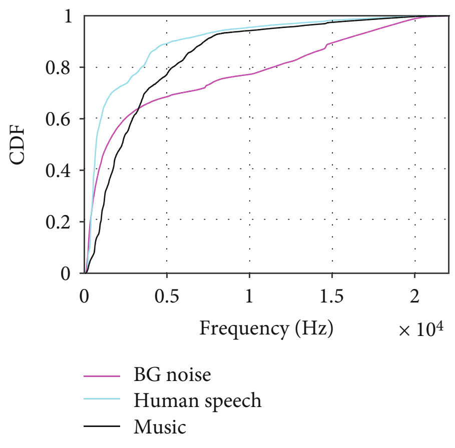
4.1.1. Audio Preprocessing. The aim of preprocessing is to filter out band noise and improve the SNR of the audio signals of interest. Since we focus on detecting keyboard and mouse interaction events in typical office environments, we first investigate the spectrum of these events and common interference in office environments. Specifically, we recorded audio signals of six *interaction events* and three kinds of external noise including background noise, human speech, and playing music with computers, using Galaxy Note 5 at a sampling rate of $F_s = 44.1\,\text{kHz}$. Each kind of acoustic events is recorded for a total period of 5 minutes. The audio signals are framed using a 100 ms Hanning window with 50% overlap and thus produce 6000 samples for each acoustic event and external noise. Following this, we compute their respective cumulative energy distribution and obtain the results as shown in Figure 2. As is shown, the energy of events signals is centered within frequency range of [5,17] kHz, while the energy of external noises is centered below 5 kHz. As a result, to filter out-of-band noise, we apply a 3-order Butterworth bandpass filter of a pass band of [5,17] kHz. To suppress in-band noise, we further apply cepstral mean normalization (CMN) [16] on the audio frames. Figure 3 shows the spectrograms of an example audio frame after bandpass filtering and CMN, respectively.

4.1.2. Event Detection and Segmentation. After preprocessing, the next step is to detect an event as well as its start and end points. However, it is not straightforward to correctly detect events using conventional thresholding method due to complex environment. In this paper, we make use of constant false alarm rate (CFAR) [25] method combining with blind segmentation to detect start and end points of each event. Essentially, CFAR is an energy-based adaptive thresholding method which adapts threshold value according to levels of external interference. Specifically, a sliding window of width W moves along the signal sequence $S(i)$ (i is the sample index). Let us assume that the power of remaining noise follows a Gaussian distribution. The average and standard deviation of signal power at sample index i are $\mu(i)$ and $\sigma(i)$, respectively. Consequently, the average signal power within a sliding window is calculated by

$$\mu(i) = \frac{1}{W}A(i) + \left(1 - \frac{1}{W}\right)\mu(i-1), \qquad (1)$$

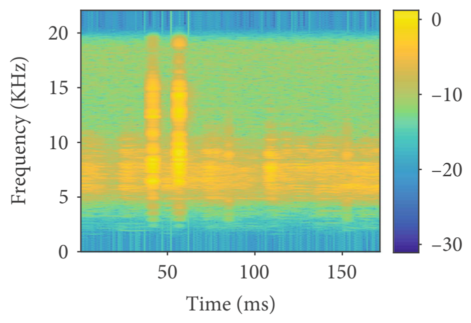


(a) The CDF of signal energy of interaction events

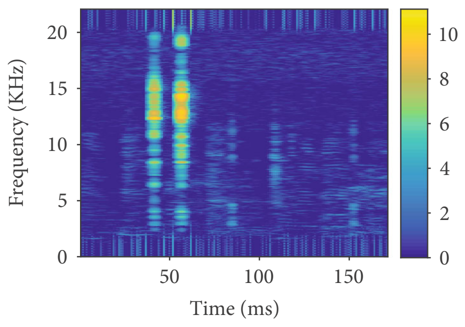


(b) The CDF of signal energy of external noises

Figure 2: The results of primary experiments which show cumulative energy distributions of *interaction events* and external noises, respectively.



(a) The spectrogram after filtering



(b) The spectrogram after CMN

Figure 3: Spectrograms of an audio frame after (a) bandpass filtering and (b) cepstral mean normalization.

where $\mu(0) = 0$ and $A(i)$ represents the cumulated power of signals in a window calculated by

$$A(i) = \frac{1}{W} \sum_{k=i}^{W+i} |S(k)|^2. \quad (2)$$

Similarly, the standard deviation at sample index i is obtained by

$$\sigma(i) = \frac{1}{W} B(i) + \left(1 - \frac{1}{W}\right) \sigma(i-1), \quad (3)$$

where $B(i)$ represents the overall standard deviation of signal power within a sliding window calculated by

$$B(i) = \sqrt{\frac{1}{W} \sum_{k=i}^{W+i} \left(|S(k)|^2 - A(k)\right)^2}. \quad (4)$$

Based on the above equations, a potential start point $S(i)$ can be determined if

$$|S(i)|^2 > \mu(i) + \gamma_1 \sigma(i), \quad (5)$$

where γ_1 is a constant parameter that is independent of noise level. Similarly, an end point is detected if

$$|S(i)|^2 < \gamma_2 \bar{\mu}, \quad (6)$$

where γ_2 is also a constant parameter dependent of noise level. $\bar{\mu}$ is the average noise power when there is no input. In our design of Behavicker, γ_1 and γ_2 are empirically set to be 2 and 40, respectively. After detecting a start point $S(k)$, we blindly extract a segment of fixed sample length $L = 16000$ in the range of $[S(m), S(n)] = [k - 1000, k + 15000]$. This length corresponds to a sampling period of 36.3 ms. This value is empirically determined based on the statistics of 600 events duration as shown in Figure 4. The result indicates that the maximum event duration is less than 30 ms for all six kinds of *interaction events*. Taking variations in realistic scenarios. we slightly relax this value and set it to be $F_s \times 36.3$ ms corresponding to 16000 samples. After blind extraction, we apply CFAR algorithm on extracted segment in two directions, one from $S(m)$ to $S(n)$ to find out the first end point (fist red point from left to right in Figure 5) and the other from $S(n)$ to $S(m)$ to find out inverse start point (the second red point from left to right in Figure 5). By doing this, we can remove irrelevant samples and extract event signal more correctly. The results of event detection and extraction are displayed as well in Figure 5. In a summary, the event detection algorithm can be summarized in Algorithm 1.

4.2. The Rationale of Hierarchical Learning Scheme. After signal preprocessing, the data is fed into the hierarchical event recognition layer as shown in Figure 1. It is worthy to think over such a question: Why we need to design such a hierarchical scheme instead of directly training a single support vector machine [26] to recognize meta events? Here, we denote a single SVM classifier as single-layer scheme and hierarchical architecture as multilevel scheme. We give some explanations on the choice of multilayer event detection scheme. First, it achieves better performance compared with single-layer scheme in experiments. We test performances of two learning schemes in cross-person scenario where the model is trained and tested with data from the attacker and target, respectively. Experimental results are shown in Figure 6. As we can see, the average accuracy of recognizing interaction events in single-layer method is about 70.6% which is about 16.0% lower than that of multilayer scheme. For scrolling wheel, the recognition accuracy is especially low in single-layer method. Second, it is reasonable for multilayer scheme to perform better since features of interaction events are extracted in a more careful way. In multilayer scheme, feature extraction and model training are conducted in a group-specific way, which is helpful to boost the performance of event recognition.

4.3. Event Classification in Coarse Layer. As demonstrated above, our goal in the first layer is to divide *interaction events* into three categories. Based on the observation of signals as shown in Figure 5, we find that some interaction events have similar signal pattern. For example, the signal pattern of keystroke and clicking mouse wheel having a large fluctuation follows with a small fluctuation (Figure 5(a)). For clicking left and clicking right mouse button, a large fluctuation follows with other large fluctuation (Figure 5(b)). For scrolling mouse wheel up and down, the large fluctuation is within small fluctuation (Figure 5(c)). Therefore, we group clicking left mouse button (CLM) and clicking right mouse button (CRM) as G1, keystroke (KS), and clicking mouse wheel (CMW) together as G2, and scrolling mouse wheel up (SWU) and down (SWD) as G3. In the following, we introduce how we construct a classification model in this layer.

4.3.1. Feature Selection. As shown in Figure 5, signals of clicking events including KS, CMW, CLM, and CRM contain two parts which correspond to *press* and *release* stages, respectively. Since the features that we extracted are closely related to the press stage, it is required to pinpoint the *press* segment in a whole signal sequence. To do this, we apply CFAR method on the extracted signal sequence and find out the first endpoint which is also the endpoint of *press* stage as shown in Figure 5 marked by the first red point. For the sake of convenience, we denote the segments between start point and first endpoint as S_1 and the whole

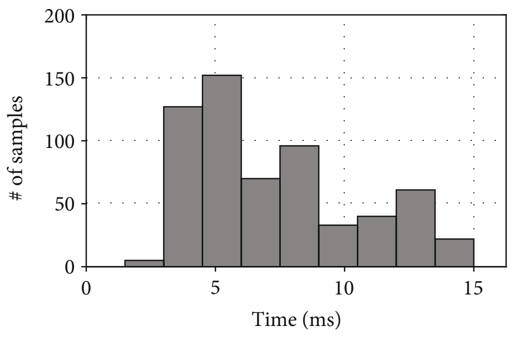


FIGURE 4: The statistics of time duration of all *interaction events* with 600 samples.

signal sequence as S. However, it is noted that for SWU and SWD, their signals last continuously and present a single stage during the whole event, which indicates that S_1 is exactly the same as S. For the convenience of narration, we let $S(i)$ with $i = 1, 2, \cdots, N$ denotes whole signal sequence of an event and $S_1(i)$ with $i = 1, 2, \cdots, n$ represents the *press* segment. We also define energy sequence $E(i)$ with $i = 1, 2, \cdots, N$ to be the square of original signal sequence. The statistics of average energy ratio of all interaction events with 600 samples is shown in Figure 7. To construct an efficient classification model, we first extract 5 features including maximum, mean, minimum, range, and standard deviation, on both original signal and energy sequences, respectively. Through this operation, we can obtain 10 features for a *interaction event* segment, denoted by S_{max}, E_{max}, S_{mean}, E_{mean}, S_{min}, E_{min}, S_{range}, E_{range}, S_{std}, and E_{std}. We also define two customized features based on our analysis on obtained signals as follows.

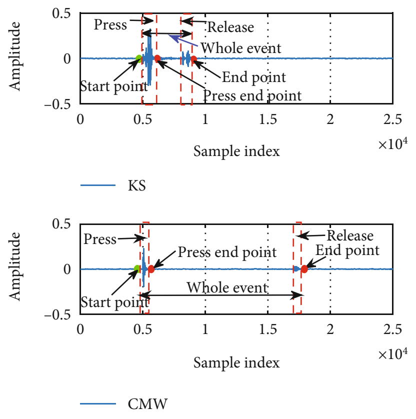
(i) Energy ratio (E_{ratio}): it describes the energy ratio of segment to the energy of a whole signal sequence. It is mathematically defined as

$$E_{\text{ratio}} = \frac{\sum_{i=1}^{n} S_1^2(i)}{\sum_{i=1}^{N} S^2(i)} \tag{7}$$

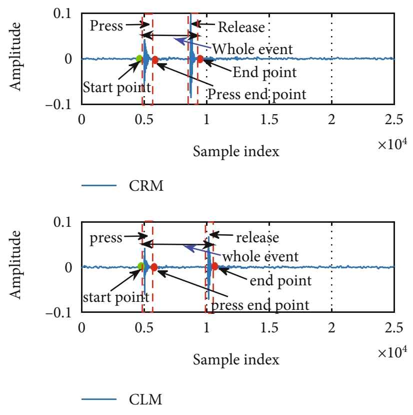
(ii) Time duration (TD): it represents the duration of signal segment between first pair of start and end point and is defined by

$$TD = \frac{1}{Fs}\left(S_1(n) - S_1(1)\right) \tag{8}$$

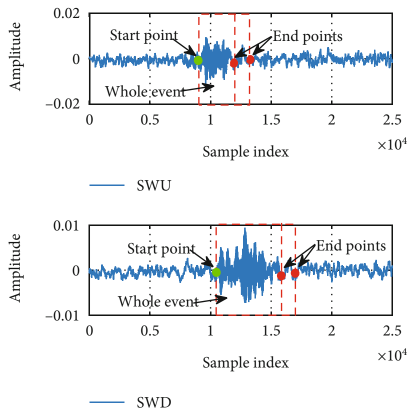
As a result, we obtain a total number of 12 features for each *interaction event* sample. For the sake of performance and efficiency, we perform feature selection with wrapper methodology in order to pick out a feature subset that helps classify *interaction events* accurately and efficiently. Generally, the wrapper methodology relies on the prediction performance of a given machine learning to assess the usefulness of feature subsets [27]. However, there are two problems to be solved in order to apply this methodology.



(a) Signals of keystroke (KS, top) and clicking mouse wheel (CMW, bottom)



(b) Signals of clicking left (CLM, top) and right mouse buttons (CRM, bottom)



(c) Signals of scrolling mouse wheel up (SWU, top) and down (SWD, bottom)

FIGURE 5: The results of event detection with modified CFAR performed on signal sequences of six basic events, namely, KS, CMW, CLM, CRM, SWU, and SWD.

```
Input:  clean audio signal sequence $S$, constant parameters $\gamma_1$ and $\gamma_2$
Output:  event segments $ES$, event start point $esp$, press(first) end point $pep$, event(release) end point $eep$,
W ⟵ FrameSig(S); framing the signal as windows sequence $W$ with step equal to 1;
while $S(++i) \neq NULL$ do
    update $\mu(i)$, $\delta(i)$ and $\bar{\mu}$, based on Equation(1)~Equation (4);
    if $|S(i)|^2 > \mu(i) + \gamma_1\delta(i)$ then
        $sig = S(i - 1000 : i + 15000)$; //Blindly segment the signal into equal length as signal $sig$;
        forward to find $pep$ that satisfied $|S(pep)|^2 < \gamma_2\bar{\mu}$;
        $sig = reverse(sig)$;
        find inverse start point as $eep$ on the reversed signal sig
        ES = S(i-1000:eep+1000);
        return $ES$, $esp$, $pep$;
    end
end
```

ALGORITHM 1: Event detection.

For one thing, searching the global optimal feature subset is a NP-hard problem and becomes quickly computationally intractable when the number of candidates features increases. In our case, there are a total number of $2^{12} - 1$ possible feature combinations which make it extremely exhausting to test each of them. To handle this, we apply a forward selection strategy in which features are progressively incorporated into increasingly larger subsets until all the features are added. The other problem is to choose an appropriate machine learning method to assess the effectiveness of feature combinations. As we utilize SVM as our classification model, we take it as the black box embedded in wrapper

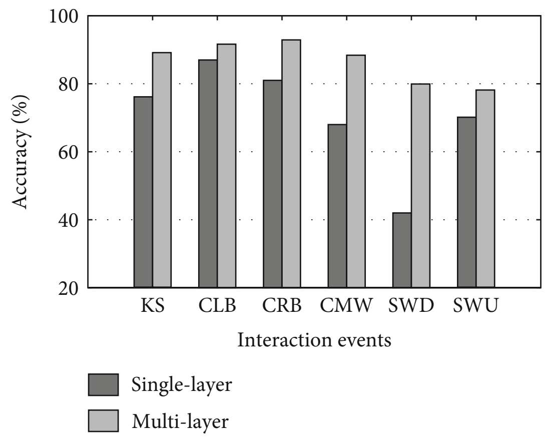


Figure 6: Performance comparison of single and multilayer schemes.

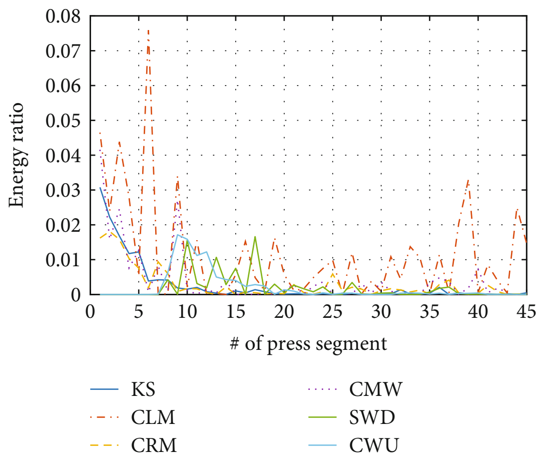


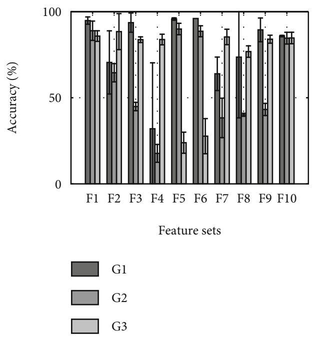
Figure 7: The statistics of average energy ratio of all *interaction events* with 600 samples.

methodology. As a result, our feature selection in coarse layer works as follows. For the total samples after preprocessing on raw data, we run forward selection method and test classification performance on each feature subset with 10-fold cross-validation. This process halts when all features are incorporated into a whole set. To reduce bias, we randomly run this process for 5 times and integrate all results to perform final feature selection. We have displayed the top 10 subsets with highest accuracies in feature selection process in Figure 8(a). As we can see, the feature subset F1 consisting of E_{std}, E_{ratio}, and TD achieves optimal performance considering accuracy and efficiency (i.e., number of features). For feature subsets achieving comparable accuracies (i.e., less than 3% difference), we give preference to the one with less features; otherwise, we choose the one with high accuracy. To visualize the effectiveness of selected features, we plot data points of events in the feature space as shown in Figure 9. It is clear that selected features can effectively partition the data points into three groups with hyperplanes.

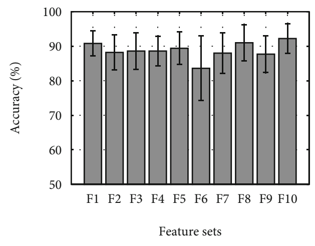
4.3.2. Model Training. After feature extraction, we train a SVM model with radial basis function (RBF) kernel using samples collected from the attacker. Compared with other classification models, SVM outperforms with its simplicity and lightweight computational overhead. As a result, it is widely used in classification, regression, and other learning tasks. In Behavicker, we make use of libSVM library and the implementation of "one-vs.-one" (OvO) approach for multiclass classification. To be noted, it is also feasible to implement "one-vs.-all" (OvA) since it achieves comparable performance with OvO. In OvO scheme, for k classes, total $(k \times (k-1))/2$ classifiers need to be trained, those trained classifiers would be used to predict the test data, and the test samples would be classified into the class with maximum votes. In order to obtain optimal performance, we carefully tune critical parameters of RBF kernel including C and γ by performing a grid search within ranges of $2^{-4} \leq C \leq 2^{12}$ and $2^{-8} \leq \gamma \leq 2^{4}$. C controls the cost of misclassification on the training data and γ can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. The parameter tuning results are shown in Figure 10.

4.4. Intragroup Event Classification. After grouping *interaction events* in coarse layer, we following conduct intragroup events classification, e.g., recognizing events individually within a group. To achieve this, we follow a similar routine with events classification in coarse layer, that is, feature selection and model training. But it is slightly different from previous stage that these two measures are conducted specifically for each group, since *interaction events* within different groups possess unique characteristics. In a whole, we first extract a universal set of features which includes Mel-frequency cepstral coefficients (*MFCCs*), average of top k RMSs (*ATR*), λth-percentile spectral rolloff (*SR*), spectral flux (*SF*), spectral centroid (*SC*), spectral entropy (*SE*), and autocorrelation coefficients (*AC*). Our considerations about extracting these features are twofold. First, they reflect characteristics of the signal from different domains including time domain, frequency domain, and cepstrum domain. As a result, they can be utilized to do classification effectively. Second, these features have been widely utilized for acoustic signals-based classification problems and proved to be effective. We borrow the idea from previous works and adjust it to fit for our problem. Due to limited space, we omit the detailed definitions and explanations of these features here and kindly refer interested readers to literature [18].

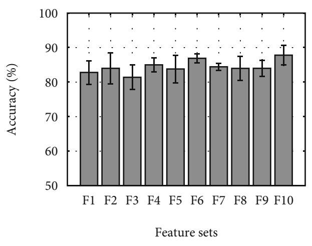
After feature extraction, we follow the same feature selection methodology as described in coarse-level event classification and obtain classification accuracies of three groups with different feature subsets. The results are shown in Figures 8(b)–8(d)for G1, G2, and G3, respectively. We can obtain several valuable observations from the results. First, different groups have different optimal feature subsets, namely, G1 with F10 consisting of MFCCs and ATR, G2 with F10 consisting of MFCCs, SR, and SF, and G3 with F9 consisting of MFCC and SC. The underlying reason is that acoustic events belonging to different groups are produced from separate mechanisms. In order to distinguish them with optimal performance, event-specific features are



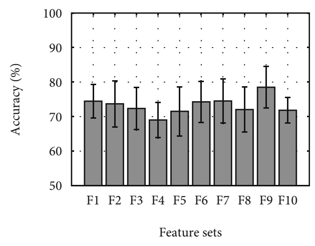
(a) Top 10 subsets of features and their corresponding accuracies in coarse layer



(b) Top 10 subsets of features and corresponding accuracies in G1



(c) Top 10 subsets of features and corresponding accuracies in G2



(d) Top 10 subsets of features and corresponding accuracies in G3

FIGURE 8: The classification accuracies of different feature subsets during feature selection process at different layers and for different groups. It is noted that features contained in subsets from F1 to F10 vary in different subfigures.

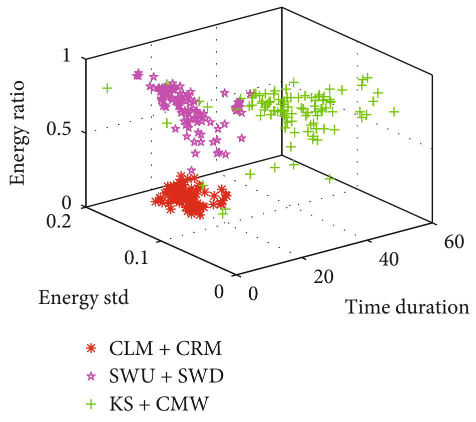


FIGURE 9: The distribution of data points in the selected feature space.

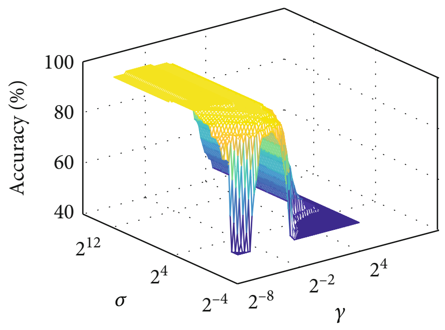


FIGURE 10: An example of parameter tuning in SVM model.

preferred. Second, different groups possess varied accuracies, among which G3 achieves the lowest accuracy. Intuitively, this is because the energy of signals produced by SWU and SWD are relatively small which makes the signals easy to be interfered by noises and hard to be differentiated. This will be verified and further explained in Evaluation section. After feature selection, we train a SVM model and tune parameters for each group following the same routine as aforementioned.

4.5. Model Adaptation. As aforementioned, a main challenge of Behavicker is that we cannot obtain labeled training data from the eavesdropping target. An alternative way to solve this problem is to train a learning model with samples collected from the attacker itself. However, due to personalized styles of using mouse and keyboard, the distribution of data points in feature space varies from person to person. As a result, the model trained with the attacker's data is not powerful enough to predict activities of a target. To get a clear idea about this, we have conducted experiments for comparison in two different scenarios, namely, identical-person and cross-person scenarios. In the former case, *interaction events* are recognized with a SVM model trained and tested on the same person. In contrast, events are recognized with the model trained and tested on data of different persons in the cross-person scenario. The experimental results are shown in Figure 11. As we can see, the average accuracy of event recognition in cross-person scenario is only about 82.3% which is about 5.4% lower than that of identical scenario. Due to the dependence of behavior recovering in the following step, it is required to further improve the performance.

To achieve this goal, we subtly design an iterative model adaptation scheme in the learning process which is similar to incremental learning technique in machine learning. As introduced in Overview, the key idea of model adaptation is to feedback samples of target with labels assigned by classification procedure to the original training data set. However, a critical problem is how we can obtain the exact labels of samples without target's active involvement, since labels assigned by classification model are with uncertainty. To resolve this problem, we utilize a metric of *confidence* which implies the certainty of classification result for each sample to help select feedback samples. Specifically, we select samples classified with high confidence to feedback to the original training data set. The rationale is based on such an insight that although people show varied patterns of performing mouse and keystroke events, there still exist common patterns as indicated by the results in cross-person scenario in Figure 11. As a result, samples classified in cross-person scenario with high confidence indicate exact categories with high probability. Mathematically, the confidence score is determined by the distance between a sample point and the hyperplane as follows [28]:

$$\Pr(\text{class} \mid \text{input}) = \Pr(y = 1 \mid x)(y = 1|x) \approx P_{A,B}(f(\mathbf{x})) = \frac{1}{1 + \exp\left(Af(\mathbf{x}) + B\right)}, \quad (9)$$

where $f(\mathbf{x}) = h(\mathbf{x}) + b$ represents the decision function of a SVM model. By determining parameters of A and B, it is capable of obtaining the confidence score of each sample after its label is assigned by the model. In the implementation of Behavicker, we feedback classified samples with confidence score above a certain threshold δ ($\delta = 0.70$ in Behavicker) and delete the same number of samples at the same time. After that, the model is retrained on the updated data set. To reduce overhead, the model adaptation procedure is performed by batch of samples. In Behavicker, we update the model with a collection of 10 samples each time. Figure 11 also shows the effect of model adaptation. It is clear that the accuracy of *interaction event* recognition is nearly the same as that of identical-person scenario after model adaptation, which indicates the effectiveness of this technique.

5. Computer-Usage Recognition

The previous parts enable us to obtain an *interaction event* sequence, which can be denoted by $L_{\text{inf}} = <e_1, e_2, \cdots, e_i, \cdots>$, where e_i represents a certain keyboard and mouse event identified in Acoustic-Based Interaction Event Recognition. Considering a certain time resolution Δt, the sequence L_{inf} reduces to $L = <e_1, e_2, \cdots, e_i, \cdots, e_n>$. This section introduces how Behavicker recognizes the computer-usage activities defined in Overview from such an event sequence L.

We propose to apply a tree-structured classifier to recognize different computer-usage activities from interaction events with keyboard and mouse. Despite its simplicity, a tree-structured classifier is able to model and interpret the differences in the interaction logics of various computer software, which is the core idea to distinguish computer-usage activities from keyboard and mouse interaction events. At different tree levels, we use different metrics to differ computer-usage activities. Figure 12 illustrates the tree structure of the classifier as well as the metrics and rules at each tree level. Before going deep into how it works, we introduce some notations by referring to Figure 12.

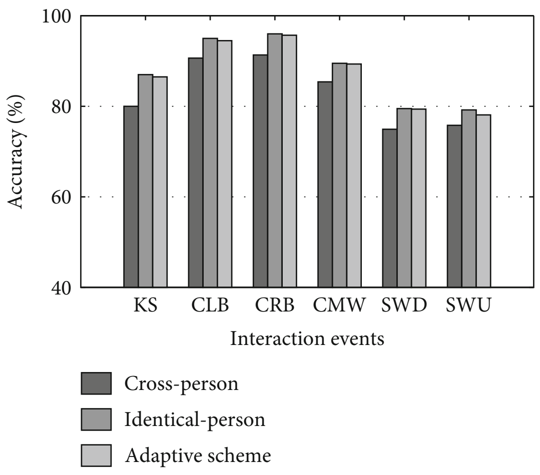


Figure 11: Performance comparison of cross- and identical-person scenarios.

(i) G_j^i: the jth activity group (i.e., rectangles in Figure 12) at the ith layer from left to right

(ii) R_j^i: the jth rule at the ith layer from left to right

(iii) G_1^1 represents the considered event sequence within time resolution Δt

Specifically, we first categorize the 7 computer-usage activities with 18 software as either *keyboard-dominant* or *mouse-dominant*. We use the ratio between frequencies of keyboard and mouse events (*KMR*) as the metric. Figure 13(a) shows the KMRs in different computer-usage activities with software.

For example, text processing usually involves more keyboard interaction events (i.e. keystrokes) than online reading. Then, we differ computer-usage activities based on the change frequency of interaction events, calculated by the event switch rate (*ESR*). Given a user interaction event sequence $L = <e_1, e_2, \cdots, e_n>$, *ESR* is defined as mean of the absolute ΔL which describes the change rate of *interaction events*. Figure 13(b) shows statistics of *ESRs* of different computer-usage activities. For instance, text processing, e.g., is likely to involve mainly keystrokes while online shopping may be mixed with keystrokes (e.g., input keywords to search products) and mouse events (e.g., clicks to select products and scrolls to go through product details). Finally, we further differentiate computer-usage activities according to temporal features including the mean duration of actions within the time window (MDA), where an action is a cluster of events with consecutive gap smaller than 10 s, and the gap between events (GBE), which is the mean of time intervals between consecutive interaction events.

Figure 13(c) illustrates the differences in MDA and GBE with examples. For example, chatting with sellers on Jingdong has lower MDA than chatting on Weixin since users

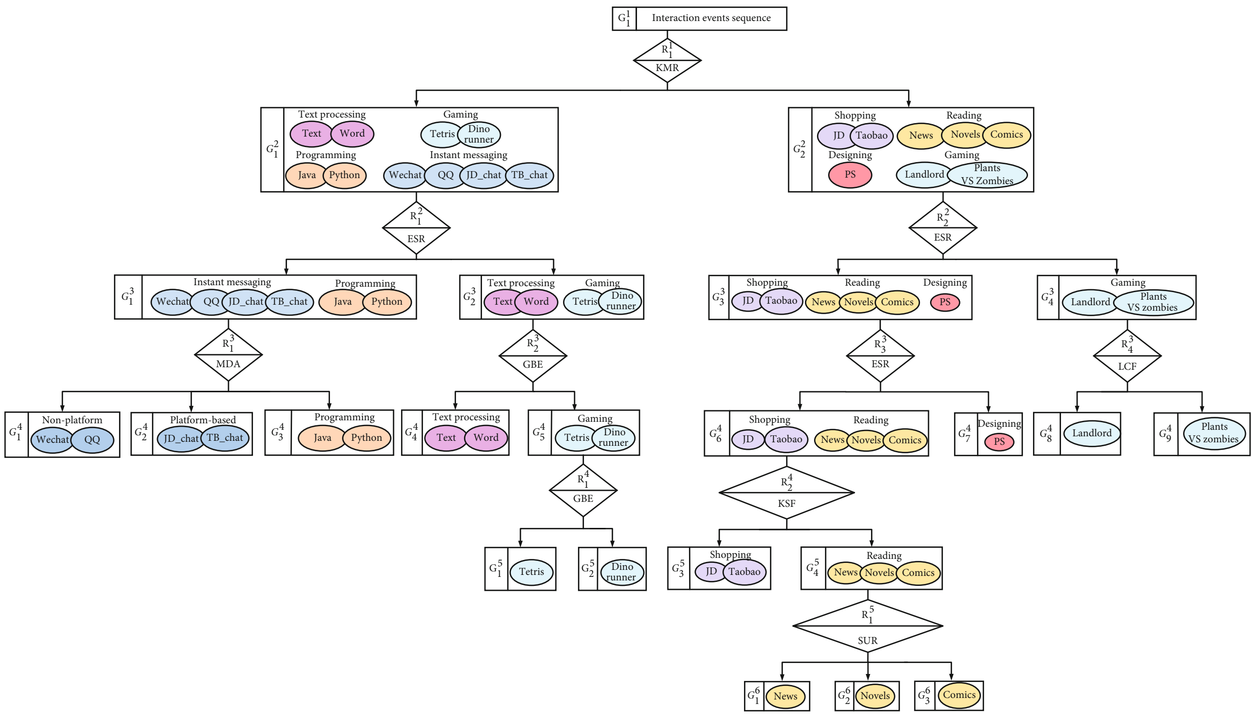
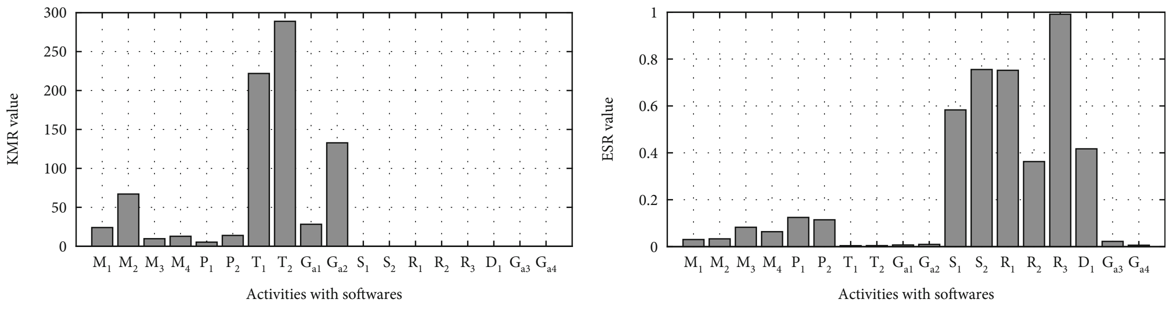
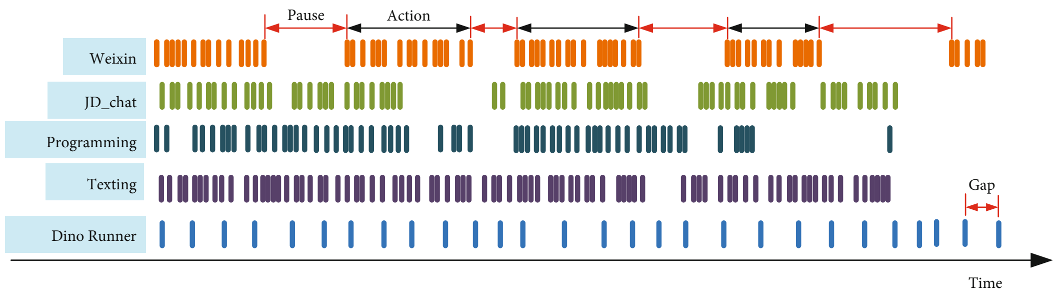


FIGURE 12: Tree-structured classifier for computer-usage recognition.



(a) The KMR values of different activities with software

(b) The ESR values of different activities with software



(c) The ESR values of different activities with software

FIGURE 13: Statistics of (a) KMR and (b) ESR for different computer-usage activities with software and the illustrations of MDA and GBE for different computer-usage activities.

need to wait longer for response in the former cases. Text processing has smaller GBE than Diano Runner. We also use left click frequency (LCF) which is defined as the number of clicking left mouse over the considered time duration, keystroke frequency (KSF) which is defined as the number of keystrokes over the considered time duration, and scrolling up rate (SUR) which is defined as the ratio of scrolling up.

We apply the following method to determine the thresholds of the rules at each branches. To decide a threshold for a metric at a rule node (i.e., diamond in Figure 12), we first obtain the distributions of 5 samples belonging to the children nodes on this metric. Then, the threshold is empirically determined by averaging the first quartile of the distribution of one child and the third quartile of the distribution of the

Table 1: Pairs of keyboards and mice for evaluation.

Pair	Keyboard			Mouse	
ID item	Brand	Model	Type	Brand	Model
K1	Lenovo	LXB-JME7155P	Membrane	Lenovo	MOEUUOA
K2	Logitech	G610	Mechanical	Logitech	M150
K3	USCorsair	STRAFE	Mechanical	Logitech	M212
K4	Lenovo	sk-8813	Membrane	Microsoft	Sculpt Ergonomic
K5	HP	KU-1156	Membrane	HP	MSU1465

other child. Afterwards, the threshold is fine-tuned using the training data set. For example, to determine *ESR* R_2^2, we first compute *ESR* distributions of samples in G_3^3 and G_4^3. Then, we average the first quartile of G_3^3 and the third quartile of G_3^3 as initial threshold and fine tune it to select an optimal one. It is to be noted that, since thresholds here reveal characteristic of distribution patterns of *interaction events*, they are more independent with users and scalable to different behavior performers. As a result, thresholds determined with attackers' samples can be easily applied for recognizing users' behaviors. What is more, due to this property, it does not need intensive data collection for determining thresholds.

6. Evaluation

6.1. Experiment Setup

6.1.1. Implementation. We prototype Behavicker on a smartphone and a back-end server. The smartphone collects acoustic signals via its built-in microphone, preprocesses the audio, and detects user interaction events as in Interaction Event Detection.

The back-end server is responsible for user interaction event recognition (Acoustic-Based Interaction Event Recognition) as well as computer-usage activity recognition (Computer-Usage Recognition). The smartphone and the back-end server are connected by Wi-Fi. We use a Samsung Galaxy Note 5 running Android OS 6.0.1 for acoustic sensing. A sampling thread continuously collects 16-bit mono audio samples at a frequency of 44.1 kHz. It feeds the audio samples to another thread for preprocessing and event detection once 17,640 samples (corresponding to 400 ms) are gathered. Audio frames without events are discarded. Frames with events detected are buffered and transmitted to the server through Wi-Fi. All processing threads are implemented in Java. We use a desktop with four-core Intel (R) Xeon (R) E3-1231 CPU and 32 G RAM running Ubuntu 16.04 LTS as the server. User interaction event recognition and computer-usage activity recognition are implemented in MATLAB. Classifiers are constructed and trained based on the libSVM library [29].

6.1.2. Data Collection. We evaluate Behavicker on data collected by different participants, devices, and office environments. Specifically, we recruit 20 participants (labeled as V_1~V_20, 12 males and 8 females) from our department to collect computer-usage activity data. To evaluate the impact of device diversity, we collect data using 5 keyboard-mouse combinations as listed in Table 1. Since we mainly target at office environments, we conduct experiments in three typical settings.

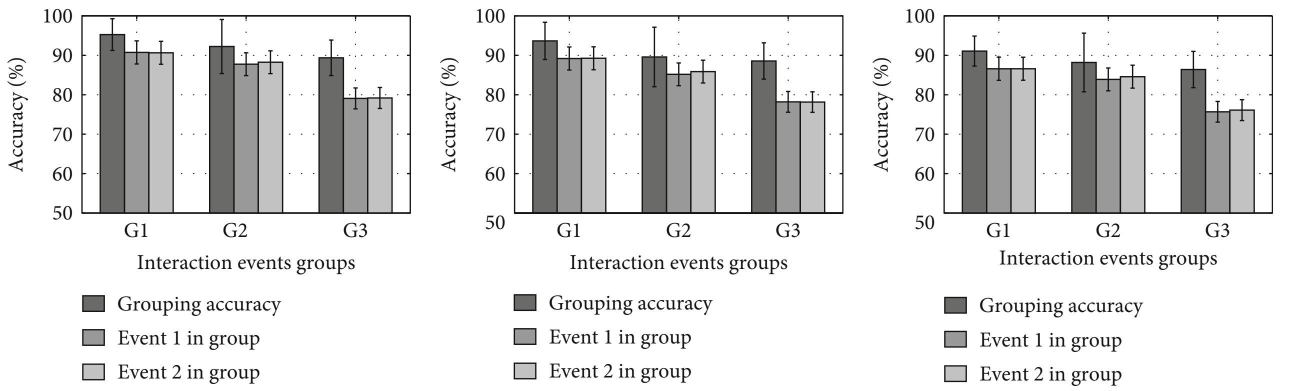
(i) Private office with noise (i.e., *setting 1*): this is a relatively quiet setting with background noise of roughly 35 dB generated by air conditioners

(ii) Public lab (i.e., *setting 2*): this is an uncontrolled setting where students work in a public lab. During the experiments, nonparticipants in the lab were unaware of the ongoing experiments and are free to work, chat, etc.

(iii) Private office with music (i.e., *setting 3*): this is a relatively noisy setting with music playing in moderate volume in the same private office. The measured noise level is about 60 dB

For user interaction event recognition, each participant is asked to perform each interaction (i.e., KS, CLB, CRB, CW, SDW, and SUW) for 100 times in each setting with every pair of keyboard and mouse. We use a Samsung Galaxy Note 5 (see Implementation) to record acoustic signals. The smartphone is placed 25 cm away from the keyboard and collects data in three different relative positions, i.e., left, right, and upper side. Therefore, the total number of interaction events is 108,000, i.e., 20 (participants) × 3 (settings) × 3 (relative positions) × 6 (interaction events) × 100 (repetitions). To guarantee the fidelity of data, we collect data of interaction events when the participants use their computers as usual. For example, they can choose to edit documents, play computer games, and chat online when we collect data of interaction events. For ease of labeling, data of only one kind of interaction event are collected each collection session.

For computer-usage activity recognition, each participant is asked to use the software in scope for 2 hours in each setting using keyboard-mouse pair K3 in Table 1. We use the same Samsung Galaxy Note 5 to collect data. The smartphone is placed roughly 25 cm away from the keyboard, but we do not restrict its relative position to the keyboard. In total, 2,520 hours of data are collected, i.e., 20 (participants) × 20 (activities) × 2 (hours) × 3 (settings). The ground truth labels are obtained by video recording with informed consent. The data collection process spans 3 months.



(a) In private office with noise (b) In public lab (c) In private office with playing music

Figure 14: The overall performance of Behavicker in typical scenarios, namely, with background noise, with people talking and with playing music. G1, G2, and G3 represent three groups of *interaction events*. G1 contains keystroke (event 1) and clicking wheel (event 2); G2 contains left click (event 1) and right click (event 2); G3 contains scrolling up (event 1) and scrolling down (event 2).

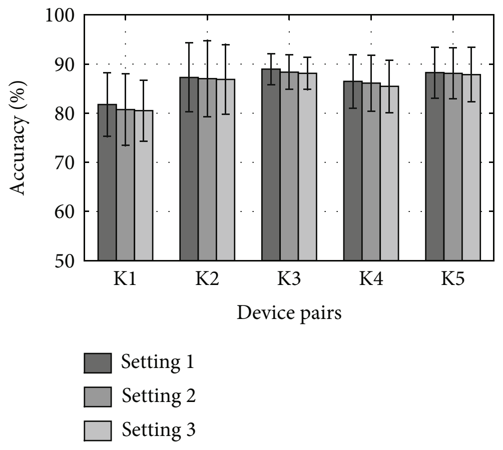


Figure 15: Performance comparison with different device pairs.

6.2. *Performance of User Interaction Event Recognition*

6.2.1. Overall Performance. Figure 14 shows the performance of Behavicker of recognizing events in group-level (i.e., coarse layer) and event-level in the three settings as aforementioned. As we can see, the group-level classification accuracies are about 95.3%, 92.2%, and 89.8% in different settings, respectively, while the average event recognition accuracies are 88.6%, 86.2%, and 84.9%. It is noted that the event-level accuracy (i.e., accuracy of recognizing *interaction events*) is obtained by multiplying group-level accuracy with intragroup classification accuracy. From the results, we can obtain several observations about the performance of Behavicker. First, Behavicker shows favorable robustness to normal noise interference. Second, compared with music, human talking shows less interference to Behavicker. The reason is that music shows higher energy in the frequency band of [4, 17] kHz compared to human speech as shown in Figure 2. As a result, there remains more noise in the denoised signals in the playing music scenario.

6.2.2. Performance of Different Hardware. To examine the scalability of Behavicker on different hardware, we compare the performance of recognizing *interaction events* with different pairs of keyboard and mouse in three scenarios. The results are shown in Figure 15. According to the results, Behavicker can achieve an average accuracy of 83.1%, 89.0%, 86.5%, 88.0%, and 88.1%, respectively. This variance is induced by the hardware diversity, which results in different acoustic signals on devices. However, since Behavicker is trained and tested on the same set of hardware, the hardware diversity has minute effect on the final performance. In the following evaluation, the results are averaged over different pairs of keyboards and mice without additional specification.

6.2.3. Performance of Different Positions. Although it is possible for attackers to collect data in the workplace of a victim, they usually tend to commit this in different positions for the sake of secrecy. Moreover, the smart device placed near the victim to collect acoustic signals will be moved to different positions during the eavesdropping process, which makes the training environment not consistent with the testing environment. To evaluate the effect of such inconsistency, we test the recognition accuracy of Behavicker with data collected from different positions. Specifically, we use one participant's data which is collected at one location of the public lab to train Behavicker, and test it with another participant's data collected in the right, front, and left of the keyboard in a private office. The corresponding results are shown in Figure 16. As we can see, even when Behavicker is trained in different positions from where it is trained, it can achieve a relatively high accuracy up to 88.3% of recognizing *interaction events*. In addition, it is noted that when the smartphone is placed in different positions, the performance of Behavicker in each position differs slightly from each other with a standard deviation of 1.2%.

6.2.4. Performance of Cross-Persons. Considering the attack scenario, it is difficult to obtain *labeled* training data from a victim for Behavicker. As a result, we train the system with data collected from one participant and test it with data from another. The scalability of Behavicker among different pairs of attackers and victims was evaluated. Since there are 20 participants in our experiments, the total number of

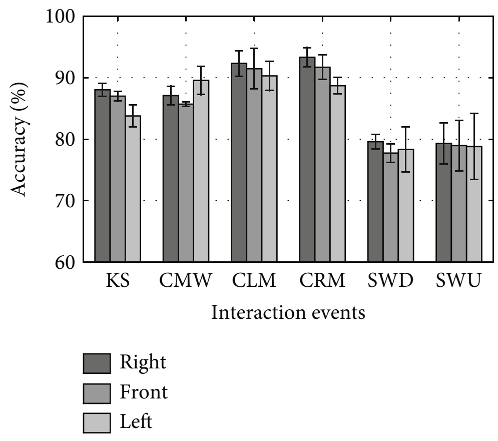


Figure 16: The performance of Behavicker in cross-position scenarios.

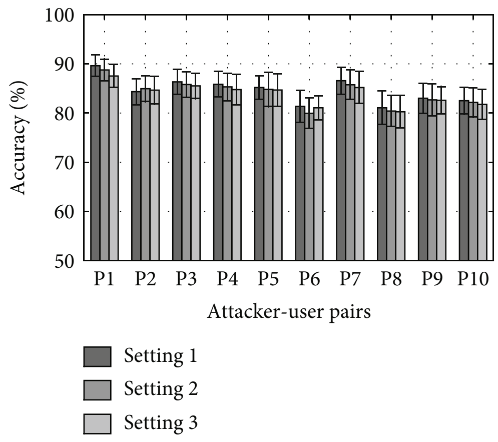


Figure 17: The performance with different attacker-user pairs.

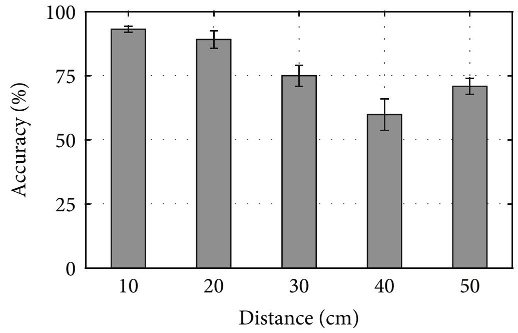


Figure 18: Performance at different recording distances.

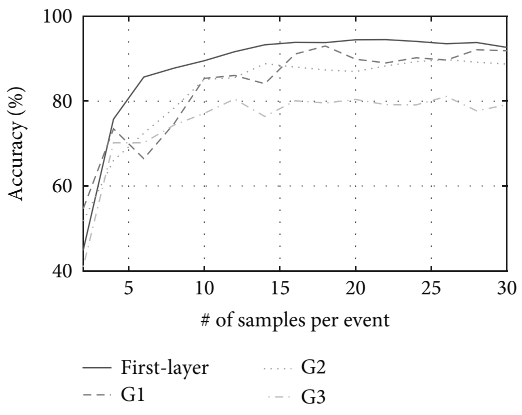


Figure 19: Performance with different number of training samples.

attacker-victim pairs is $A_{20}^{2} = 380$. Considering the minute difference in results of pairs, we randomly display the overall accuracy of 10 pairs as shown in Figure 17. The average accuracy of these pairs is 88.9%, and the corresponding standard deviation is 2.3%, which indicates that Behavicker can achieve favorable performance with different pairs of attacker and victim. The reasons for Behavicker's scalability are twofold. First, although the habits of performing *interaction events* are diverse among different persons, the features that we utilize for coarse-grained grouping and *interaction event* classification reflect the intrinsic characteristics of events. Second, the model adaptation scheme gradually updates the itself with victim's *pseudo-labeled* samples during eavesdropping, which finally evolves to be a model trained with samples of the victim.

6.2.5. Impact of Eavesdropping Distances. To evaluate the impact of the distance between recorder (i.e., smartphone) and keyboard, we conduct experiments by placing the smartphone at positions with distances of 10 cm, 20 cm, 30 cm, 40 cm, and 50 cm from the keyboard. There are two points to be pointed out. First, this set of experiments are conducted by 2 instead of 20 participants since differences caused by participants are negligible. Second, we train Behavicker with data collected at 20 cm and test it with data at five distances, respectively. Figure 18 displays accuracies averaged across both persons and device pairs at each distance. As we can see, within 30 centimeters, the accuracy of *interaction event* recognition maintains an average value of 85.6%. When the distance increases, the accuracy decreases due to lower signal to noise ratio (SNR). The abnormal case in 50 cm is caused by tapping keyboard and mouse emphatically by a participant according to our posteriori investigation.

6.2.6. Impact of Training Samples. Figure 19 shows the effect of training samples on the overall accuracy of recognizing *interaction events*. The evaluation result is accomplished by averaging the results of different attacker-user pairs with different combinations of positions. In this figure, we display the training overhead in the first and second layer. As we can see, for the first-layer classification, the number of training samples is 15, the average grouping accuracy in the first layer has reached about 92.6%, and the performance increases little even when the training samples increases further. This trend is very similar to the intragroup classification for three groups. This indicates that Behavicker requires reasonable training effort and is practical in real cases. The variance of recognizing *interaction events* within groups is larger than that of grouping in the first layer, even after the number of training samples exceeds 15. We think that it is reasonable since *interaction events* within a group

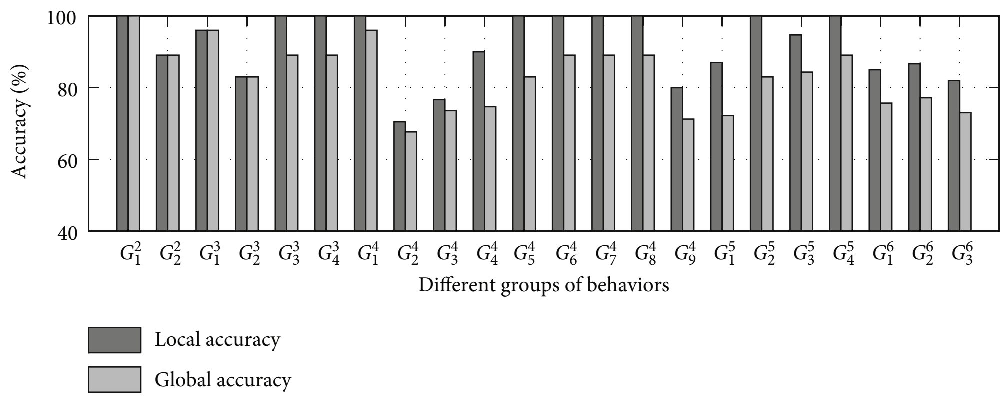


Figure 20: The local and global recognition accuracy of different activity groups at each layer.

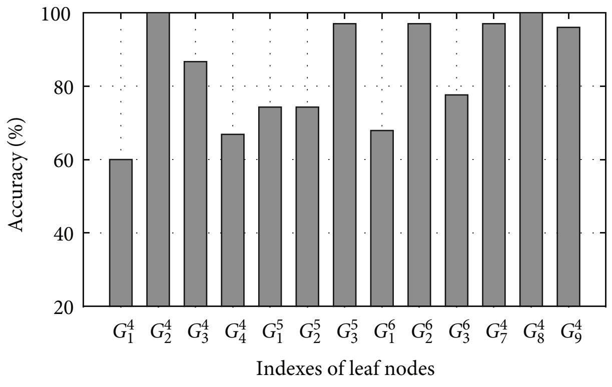


Figure 21: Global accuracies of activity recognition at leaf nodes.

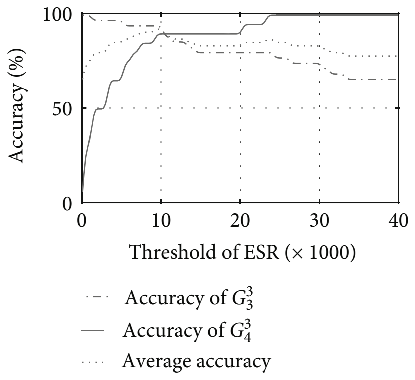


Figure 22: Performance with different thresholds of *ESR* in R_1^2.

share more similarities and are more difficult to be distinguished.

6.2.7. Impact of Model Adaptation. To evaluate the impact of model adaptation, we display the overall *interaction event* recognition accuracy varying with the number of updated samples under different confidence threshold δ. We have tested the confidence threshold δ in model adaptation from 0.65 to 0.90 with a step value of 0.05. As we can see, the threshold has great impact on the recognition accuracy of *interaction events*. When the confidence is set too high, the

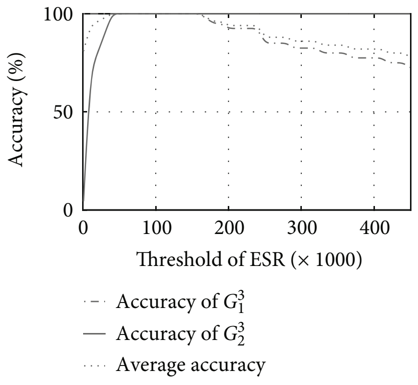
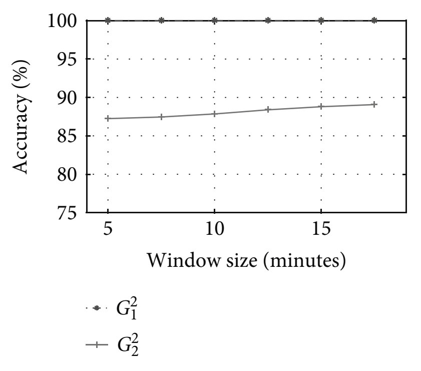


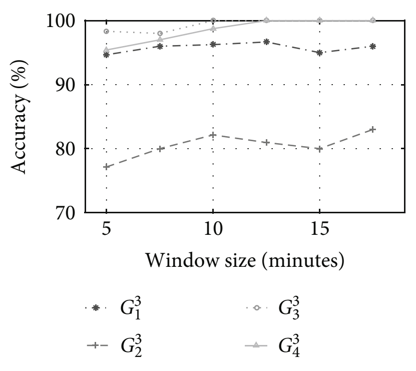
Figure 23: Performance with different thresholds of *ESR* in R_2^2.

overall accuracy decreases with the increasing number of updated samples. The larger the threshold is, the faster the accuracy decreases. The reason for this trend is that although a high confidence threshold guarantees the returned samples being correctly labeled, it filters out many other samples belonging to a certain category which makes the retrained classifier biased. As the returned samples increase, the bias of the classifier is also magnified. On the other hand, when the confidence threshold is set too low, the returned samples contain more mislabeled ones which results in errors in the retrained classifier. Considering the trade-off between rejecting mislabeled samples and keeping the completeness of training data set, a moderate confidence threshold is an optimal choice as shown in the figure. In our implementation of Behavicker, we set the confidence threshold δ to 0.7.

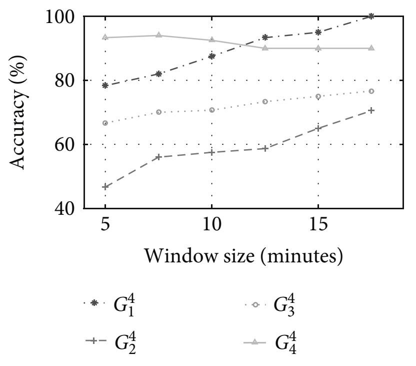
6.3. Performance of Computer-Usage Activity Recognition. Before presenting evaluation, we first introduce two concepts that will be utilized in the following. For an activity classified into a node G_j^i in Figure 12, there are two types of accuracies, namely, *local accuracy* and *global accuracy*. The *local accuracy* of an activity in node G_j^i represents the probability of its being classified into present node from parent node of G_j^i, while the *global accuracy* represents the probability of its being classified into G_j^i from the root node G_1^1. Consequently, the global accuracy is a multiplicative function of local accuracies along a same path.



(a) Local accuracy of recognizing activities at the 2nd layer with different window sizes



(b) Local accuracy of recognizing activities at the 3rd layer with different window sizes



(c) Local accuracy of recognizing activities at the 4th layer with different window sizes

FIGURE 24: The impact of time resolution on the accuracy of recognizing activities at different layers.

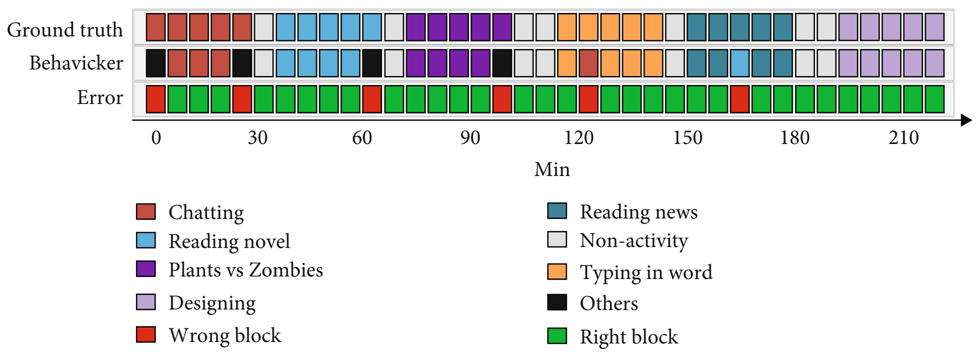


FIGURE 25: The results of our user study experiment in which a participant conducts different activities with his computer for 220 minutes.

6.3.1. Multiscale Accuracy. Since activities are recognized at multiple scales with preserving semantics, it is meaningful to evaluate performance of activity classification at different layers. Figure 20 shows local and global accuracies of recognizing activities at different layers. The average local and global accuracies are (94.5%, 94.5%), (94.8%, 89.6%), (90.8%, 82.6%), (95.1%, 82.4%), and (84.6%, 78.6%) for each layer from the top to the bottom, respectively. An overall trend is that global accuracies of recognizing activities decrease with the depth of layers, since they are obtained by multiplying local accuracies along the path from the root to present node. We can also observe that local accuracies decrease with layer depth as well, for the reason that activities belonging to the same group in deeper layers share more similarities and thus are more challenging to be differentiated.

6.3.2. The Ultimate Accuracy. The ultimate accuracy describes how accurately activities can be classified at the leaf node. In our evaluation, an original acoustic signal sequence corresponding to an activity is first analyzed by interaction event recognition methods to obtain an event sequence. After that, the whole sequence is divided into small segments by a window of 10 minutes sliding with a step of 5 minutes. As a result, we can obtain a number of segments containing interaction events. Each segment is then fed into activity recognition scheme as described in Computer-Usage Recognition. Figure 21 shows the ultimate accuracies of different activities with an average value of 82.7% which indicates the effectiveness of our scheme. Referring to Figure 12, it is noted that some activities are grouped at leaf nodes and evaluated as a whole.

6.3.3. Impact of Thresholds. Even though thresholds used in activity recognition are automatically determined by our method, we also evaluate their impacts by varying them in a certain range. As there are a number of branches in activity recognition tree, we only pick out activity recognition at layer 2 for example. Figures 22 and 23 show performance at layer 2 for G_1^2 and G_2^2, respectively. As we can see, both average accuracies increase with the thresholds of *ESR* until a certain value. When the threshold of *ESR* exceeds this value, the average accuracies decrease accordingly. It is because with different thresholds of *ESR*, accuracies of two child nodes vary differently. More interestingly, the value of the turning point is much close to the threshold chosen by our method described in Computer-Usage Recognition which verifies its effectiveness.

6.3.4. Impact of Time Resolution Δt. Time resolution of analyzing activities refers to the width of a moving window in segmenting a whole sequence of interaction events. As different lengths of an event sequence reflect different levels of activity pattern, time resolution has impact on activity recognition. We vary the window size from 2.5 minutes to 15 minutes and test the performance of activity recognition at different layers. As local accuracy better reveals the impact of time resolution, we evaluate the performance by this metric for groups in layer 2 to layer 4 here and obtain results as shown in Figure 24. As we can see, the local accuracy of an activity group in different layers increases with the window size. This is because a wider window covers a more complete pattern of considered activity and is helpful for differentiating different activities. However, when it reaches 10 minutes, further increasing does not enhance performance notably for most activities in different layers. What is more, we can notice that time resolution has larger impact on activity recognition at deep layers which demonstrates that fine-grained activity classification is more sensitive to time resolution.

6.3.5. Case Study. Finally, we conduct a user study to evaluate Behavicker in the case where different activities are conducted by a participant sequentially. During the user study, a participant is free to perform activities, each of which lasts for 30 minutes, with his/her own desktop PC. The collected acoustic signals are analyzed by a sliding window of 10 minutes with an overlap of 5 minutes. Figure 25 displays the results in which each block represents an activity with a duration of 10 minutes. As is shown, a red block indicates a wrongly recognized activity, while a green one represents a correctly recognized activity. As we can see, there are only 6 error blocks among a total number of 37 blocks, resulting in an accuracy of 83.8%. Moreover, we can observe that 4 error blocks happen at the beginning or end of an activity. This is because transitions between different activities bring about noises to the distribution pattern of interaction events of a pure activity, which is more likely to induce errors.

7. Conclusion

In this paper, we shed light on the possibility of eavesdropping common computer-usage activities through acoustic side channel, by designing and implementing a prototype system Behavicker. Different from previous works, our system infers a target's computer-usage activities at multiple scales based on *interaction event* recognition with lightweight training. To guarantee and validate its effectiveness, we propose novel data processing techniques and conduct comprehensive real-world experiments. Experimental results reveal an ignored fact that acoustic side channel enables an eavesdropper to recover a target's computer-usage behaviors with a high accuracy. We anticipate that our work can provide a warning for user if they are sensitive to behavior leakage while using computers.

Data Availability

The experimental data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

This article is extended and revised from the previously version in SSRN [30].

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported in part by the China NSFC Grant (No. 62172286) and Guangdong NSF Grant (No. 2022A1515011509).

References

[1] Gartner Inc, "User survey analysis: mobile device adoption at the workplace is not yet mature," 2016, //http://www.gartner.com/newsroom/id/3528217.

[2] T. Zhu, Q. Ma, S. Zhang, and Y. Liu, "Context-free attacks using keyboard acoustic emanations," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pp. 453–464, 2014.

[3] X. Ji, Y. Cheng, W. Xu et al., "No seeing is also believing: electromagneticemission- based application guessing attacks via smartphones," *IEEE Transactions on Mobile Computing*, 2021.

[4] J. Liu, Y. Wang, G. Kar, Y. Chen, J. Yang, and M. Gruteser, "Snooping keystrokes with mm-level audio ranging on a single phone," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 142–154, 2015.

[5] M. G. Kuhn and R. J. Anderson, "Soft tempest: hidden data transmission using electromagnetic emanations," in *International Workshop on Information Hiding*, pp. 124–142, Springer, 1998.

[6] M. Vuagnoux and S. Pasini, “Compromising electromagnetic emanations of wired and wireless keyboards,” *USENIX Security Symposium*, pp. , 20091–16, 2009.

[7] M. G. Kuhn, “Optical time-domain eavesdropping risks of CRT displays,” in *Proceedings. 2002 IEEE Symposium on Security and Privacy*, pp. 3–18, 2002.

[8] J. Loughry and D. A. Umphress, “Information leakage from optical emanations,” *ACM Transactions on Information and System Security*, vol. 5, no. 3, pp. 262–289, 2002.

[9] G. de Souza Faria and H. Y. Kim, “Identification of pressed keys from mechanical vibrations,” *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 7, pp. 1221–1229, 2013.

[10] P. Marquardt, A. Verma, H. Carter, and P. Traynor, “(sp) iPhone: decoding vibrations from nearby keyboards using mobile phone accelerometers,” in *Proceedings of ACM CCS*, pp. 551–562, 2011.

[11] D. Asonov and R. Agrawal, “Keyboard acoustic emanations,” in *Proceedings of IEEE Symposium on Security and Privacy*, pp. 3–11, 2004.

[12] Y. Berger, A. Wool, and A. Yeredor, “Dictionary attacks using keyboard acoustic emanations,” in *Proceedings of ACM CCS*, pp. 245–254, 2006.

[13] L. Zhuang, F. Zhou, J. Doug, and J. D. Tygar, “Keyboard acoustic emanations revisited,” *ACM Transactions on Information and System Security*, vol. 13, no. 1, p. 3, 2009.

[14] D. Salomon, *Elements of Computer Security*, Springer Science & Business Media, 2010.

[15] Y. Bi, M. Lv, C. Song, W. Xu, N. Guan, and W. Yi, “Autodietary: a wearable acoustic sensor system for food intake recognition in daily life,” *IEEE Sensors Journal*, vol. 16, no. 3, pp. 806–816, 2016.

[16] J. Korpela, R. Miyaji, T. Maekawa, K. Nozaki, and H. Tamagawa, “Evaluating tooth brushing performance with smartphone sound data,” in *Proceedings of ACM Ubicomp*, pp. 109–120, 2015.

[17] W. Gu, Z. Yang, L. Shangguan, W. Sun, K. Jin, and Y. Liu, “Intelligent sleep stage mining service with smartphones,” in *Proceedings of ACM Ubicomp*, pp. 649–660, 2014.

[18] X. Sun, Z. Lu, W. Hu, and G. Cao, “Symdetector: detecting sound-related respiratory symptoms using smartphones,” in *Proceedings of ACM Ubicomp*, pp. 97–108, 2015.

[19] T. Hao, G. Xing, and G. Zhou, “iSleep: unobtrusive sleep quality monitoring using smartphones,” in *Proceedings of ACM SenSys*, 2013.

[20] Y. Ren, C. Wang, J. Yang, and Y. Chen, “Fine grained sleep monitoring: hearing your breathing with smartphones,” in *Proceedings of IEEE Infocom*, pp. 1194–1202, 2015.

[21] L. Hong, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, “Soundsense: scalable sound sensing for people-centric applications on mobile phones,” in *Proceedings of the ACM MobiSys*, pp. 165–178, 2009.

[22] H. Lu, D. Frauendorfer, M. Rabbi et al., “Stresssense: detecting stress in unconstrained acoustic environments using smartphones,” in *Proceedings of ACM Ubicomp*, pp. 351–360, 2012.

[23] S. B. Moqadam, A. S. Asheghabadi, and X. Jing, “A novel hybrid approach to pattern recognition of finger movements and grasping gestures in upper limb amputees,” *IEEE Sensors Journal*, vol. 22, no. 3, pp. 2591–2602, 2022.

[24] S. Ganguli, “Computer operating systems: from every palm to the entire cosmos in the 21st century lifestyle,” *Computer Society of India Communications*, vol. 40, no. 11, pp. 5–8, 2017.

[25] M. A. Richards, *Fundamentals of Radar Signal Processing*, McGraw-Hill Education, 2014.

[26] C. Cortes and V. Vapnik, “Supportvector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[27] I. Guyon and A. Elisseeff, “An introduction to variable and feature selection,” *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[28] H.-T. Lin, C.-J. Lin, and R. C. Weng, “A note on Platt’s probabilistic outputs for support vector machines,” *Machine Learning*, vol. 68, no. 3, pp. 267–276, 2007.

[29] C.-C. Chang and C.-J. Lin, “LIBSVM-a library for support vector machines,” *ACM transactions on intelligent systems and technology*, vol. 2, no. 3, pp. 1–27, 2016, https://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[30] M. Chen, Y. Zou, and W. Kaishun, “Behavicker: eavesdropping computer-usage activities through acoustic side channel,” *SSRN Electronic Journal*, 2022, Available at SSRN 4019830.