Cross-site Input Inference Attacks on Mobile Web Users

Rui Zhao, Chuan Yue(✉), and Qi Han

Colorado School of Mines, Golden, CO, USA
{ruizhao,chuanyue,qhan}@mines.edu

Abstract. In this paper, we investigate severe cross-site input inference attacks that may compromise the security of every mobile Web user, and quantify the extent to which they can be effective. We formulate our attacks as a typical multi-class classification problem, and build an inference framework that trains a classifier in the training phase and predicts a user's new inputs in the attacking phase. To make our attacks effective and realistic, we design unique techniques, and address major data quality and data segmentation challenges. We intensively evaluate the effectiveness of our attacks using keystrokes collected from 20 participants. Overall, our attacks are effective, for example, they are about 10.8 times more effective than the random guessing attacks regarding inferring letters. Our results demonstrate that researchers, smartphone vendors, and app developers should pay serious attention to the severe cross-site input inference attacks that can be pervasively performed, and should start to design and deploy effective defense techniques.

Keywords: Mobile · Web · Cross-site input inference · Motion sensor

1 Introduction

Smartphones have been severely targeted by cybercrimes, and their sensors have created many new vulnerabilities for attackers to compromise users' security and privacy. One typical vulnerability is that high-resolution motion sensors, such as accelerometer and gyroscope, could be used as side channels for attackers to infer users' sensitive keyboard tappings on smartphones. Such *input inference attacks* are feasible because motion sensor data are often correlated to the tapping behaviors of users and the positions of keys on a keyboard.

Some researchers have studied the effectiveness of input inference attacks performed by malicious native apps on smartphones, but their threat models and focuses are completely different from ours, and their attacks are not as challenging as ours (Sect. 2). While input inference attacks can be performed by malicious native apps, they can indeed be more ***pervasively performed*** by malicious webpages to cause even ***severer consequences*** to *mobile Web users* [8] who interact with webpages through either mobile browsers or WebView components of native apps. On both iOS and Android platforms, JavaScript code

X. Lin et al. (Eds.): SecureComm 2017, LNICST 238, pp. 629–643, 2018.
https://doi.org/10.1007/978-3-319-78813-5_32

on regular webpages can register to receive device motion events and access motion sensor data. This access does not require a user to explicitly grant any permission, install any software, or perform any configuration, and it can even be performed cross sites to create ***a powerful side channel to bypass the fundamental Same Origin Policy*** [9] that protects the Web [8].

In this paper, we investigate such severe cross-site input inference attacks and quantify their effectiveness. We formulate our attacks as a typical multi-class classification problem, and build an inference framework that takes the supervised machine learning approach to train a classifier in the training phase for predicting a user's new inputs in the attacking phase. However, two major challenges need to be addressed to make our attacks effective and realistic. The first is on ***data quality***, i.e., the quality of the collected motion sensor data for certain keystrokes could be low. The second is on ***data segmentation***, i.e., the key down and up events cannot be obtained in the attacking phase to accurately segment motion sensor data for individual keystrokes because cross-site (or origin) collection of key events is prohibited by the Same Origin Policy [9].

To address the data quality challenge, we designed two main techniques: *training data screening* and *fine-grained data filtering*. To address the data segmentation challenge, we designed a *key down timestamp detection and adjustment* technique. To evaluate the effectiveness of our cross-site input inference attacks, we collected keystrokes on 26 letters, 10 digits, and 3 special characters from 20 participants. On average, our attacks achieved 38.83%, 50.79%, and 31.36% inference accuracy (based on F-measure scores) on three charsets *lowercase letters*, *digits together with special characters*, and *all the 39 characters*, respectively. Intuitively, on the letter charset, our attacks are about 10.8 times more effective than the random guessing attacks. Our training data screening technique improved the inference accuracy against all participants by 8.03%, 9.93%, and 7.21% on the three charsets, respectively; our fine-grained data filtering technique improved the inference accuracy against the majority of participants by 1.14%, 1.76%, and 1.27% on the three charsets, respectively. Our key down timestamp detection and adjustment technique achieved 86.32% accuracy on keystroke data segmentation.

2 Threat Model and Related Work

The basic threat model in our attacks is that malicious JavaScript code can collect smartphone motion sensor data and train a machine learning classifier to infer a user's sensitive inputs cross websites, thus bypassing the security protection of Same-Origin Policy [9]. Especially, two types of cross-site input inference attacks, *parent-to-child* and *child-to-parent*, can occur as proposed by Yue [8]. In the *parent-to-child cross-site input inference attacks*, a parent document collects motion sensor data to infer users' sensitive inputs in a child (e.g., *iframe*) document [8]. In the *child-to-parent cross-site input inference attacks*, a child document collects motion sensor data to infer users' sensitive inputs in a parent document [8]. On both iOS and Android platforms, these attacks do not require

a user to explicitly grant any permission, install any software, or perform any configuration. Collecting training data is feasible because attackers can trick a user to type specific (i.e., labeled) non-sensitive inputs on their webpages – attackers can collect the motion sensor data as well as the corresponding key down and up events from the same webpages to accurately segment these data.

Some researchers have studied the effectiveness of input inference attacks on smartphones. However, the threat models and focuses of the existing efforts are different from ours, and their attacks are not as challenging as ours. First, they mainly focused on investigating the attacks performed by the native apps [1,2,7], and assumed that malicious apps have been installed on users' smartphones to access the motion sensor data. Second, they mainly focused on investigating the attacks that target at touchscreen lock PINs [1,7], which could be valuable only if they are reused by smartphone owners on online services or if the smartphone itself is stolen. Third, they often used apps' built-in keyboards [1,7] and/or large digit-only keyboards [1,7] to collect motion sensor data and perform experiments, and did not study the attack effectiveness using real alphanumeric keyboards. Fourth, they often collected the key down and up events to accurately segment motion sensor data (i.e., identifying the start and end time) to infer individual keystrokes [1,7]; however, in reality smartphone platforms do not allow the cross-app collection of key events for security reasons.

3 Design of Cross-site Input Inference Attacks

3.1 Overview of the Framework

We formulate our attacks as a typical multi-class classification problem, and build a framework that takes the supervised machine learning approach to train a classifier in the training phase for inferring a user's new inputs in the attacking phase as shown in Fig. 1. The framework consists of six components. The *sensor data segmentation* component segments motion sensor data for individual keystrokes. The *training data screening* component calculates the character-specific quality scores for individual keystrokes and selects the motion sensor data of good-quality keystrokes into the training dataset. The *fine-grained data filtering* component selects user-specific frequency bands with varying lengths for reducing the noise in the motion sensor data. The *feature extraction* component statistically derives both time-domain and frequency-domain features from the filtered motion sensor data. The *model training* component trains a machine learning classifier from the extracted features. The *prediction* component uses the trained classifier to predict new characters tapped by a user.

In the training phase, attackers are capable of using JavaScript code to collect both motion sensor data and *key events* (i.e., key down and up) at the client side on a user's smartphone; these data are then sent to an attacker's server, and further segmented, screened, and filtered for extracting features to train a classifier. By leveraging the corresponding key events for identifying the start and end time, this motion sensor data segmentation for individual keystrokes in the training phase can be accurately performed. By selecting the motion sensor

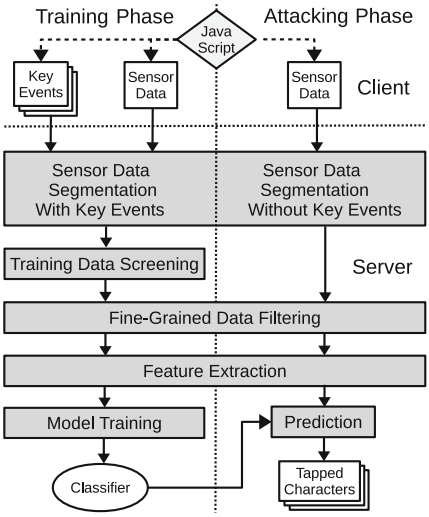


Fig. 1. The framework for cross-site input inference attacks

data of good-quality keystrokes and by further filtering out the noise at a fine granularity, the classifier can be better trained for performing the attacks.

In the attacking phase, attackers are only capable of collecting motion sensor data because cross-site (or cross-origin) collection of key events is prohibited by the Same Origin Policy; the motion sensor data are then sent to the attacker's server, and further segmented and filtered for extracting features to predict the tapped characters using the trained classifier. Due to the lack of key events in the attacking phase, accurate motion sensor data segmentation becomes very challenging and an effective technique must be designed. Character-specific quality scores cannot be calculated in the attacking phase because the tapped characters are unknown and are indeed the targets of the inference attacks. Therefore, our framework currently does not include data screening in the attacking phase.

3.2 Motion Sensor Data Segmentation

Figure 2 illustrates the algorithms used for sensor data segmentation in the two phases. The *Identify-Keystroke-TimeWindows* subroutine accepts a sequence of key down timestamps T as the input and returns a sequence of keystroke time windows W as the output. For each key down timestamp T_j, the timestamps $T_j - \mathit{offset\_start}$ and $T_j + \mathit{offset\_end}$ are identified as the start and end of the corresponding keystroke time window, respectively. This time window identification method has been commonly used by researchers in input inference attacks [1,3,7]. They often use 100 and 150 ms as the values of *offset_start* and *offset_end*, respectively, based on their observations on the time relationship between motion sensor data and key events; we have the similar observation, and thus used the same offset values in this subroutine.

```
// S = (S_{t_1}, S_{t_2}, ··· , S_{t_n}): motion sensor data from time t_1 to t_n
// S_{t_i} = (x_{t_i}, y_{t_i}, z_{t_i}, α_{t_i}, β_{t_i}, γ_{t_i}): motion sensor data at time t_i, where x_{t_i}, y_{t_i}, z_{t_i} represent
      acceleration forces on axes x, y, z, and α_{t_i}, β_{t_i}, γ_{t_i} represent rotation rates on axes z, x, y
// T = (T_1, T_2, ··· , T_m): a sequence of m key down timestamps
// W = (W_1, W_2, ··· , W_m): a sequence of m identified time windows,
      where W_i = (W_i^S, W_i^E) represents the start and end time of a window

Segment-SensorData-With-KeyEvents (T) // Used in the training phase
1    W = Identify-Keystroke-TimeWindows (T)
2    W = Adjust-Keystroke-TimeWindows (W)
3    return W

Segment-SensorData-Without-KeyEvents (S) // Used in the attacking phase
1    T = Detect-KeyDown-Timestamps (S)
2    W = Identify-Keystroke-TimeWindows (T)
3    W = Adjust-Keystroke-TimeWindows (W)
4    return W

Detect-KeyDown-Timestamps (S)
1    S = Filter-Data (S, start_frequency, end_frequency)
2    M^A = M^R = () // Magnitude for acceleration forces and rotation rates
3    for t in t_1 : t_n
4         M_t^A = sqrt(x_t^2 + y_t^2 + z_t^2); M_t^R = sqrt(α_t^2 + β_t^2 + γ_t^2)
5    T^A = Find-Peak-Timestamps (M^A); T^R = Find-Peak-Timestamps (M^R)
6    T = Merge-Peak-Timestamps (T^A, T^R)
7    return T

Identify-Keystroke-TimeWindows (T)
1    for j in 1 : m
2         W_j^S = T_j − offset_start; W_j^E = T_j + offset_end
3    return W

Adjust-Keystroke-TimeWindows (W)
1    for j in 1 : m − 1
2         overlap = W_j^E − W_{j+1}^S // Overlap between two keystrokes
3         if overlap ≤ 0 // No overlap
4              // Do nothing
5         else if overlap > ((W_{j+1}^S + offset_start)−
               (W_j^E − offset_end)) × overlap_threshold // Heavy overlap
6              mark W_j and W_{j+1} as heavily overlapped time windows
7         else // Slight overlap, split the overlapped region
8              W_j^E = W_j^E − overlap/2; W_{j+1}^S = W_{j+1}^S + overlap/2
9    remove the marked heavily overlapped time windows from W
10   return W
```

Fig. 2. Sensor data segmentation algorithms in the two phases

The *Detect-KeyDown-Timestamps* subroutine accepts the motion sensor data S from timestamp t_1 to timestamp t_n as the input, finds their peak values, and returns a sequence of key down timestamps T as the output. The subroutine first applies a band filter from $start\_frequency$ to $end\_frequency$ on the sensor data S at line 1. Because the peak values of sensor data are often well captured by their high frequency components, using a filter with a high-pass band (e.g., from 10 Hz to 30 Hz in our case) here can help us accurately detect the key down timestamps. To comprehensively consider acceleration forces and rotation rates along all the three axes, the subroutine computes the Euclidean magnitude values M_t^A (for acceleration forces) and M_t^R (for rotation rates) at line 4 for each timestamp t. At line 5, the peak values in M^A and M^R are identified using a

sliding window based on the average keystroke duration observed in the training data, and their timestamps are saved to the sequences, T^A and T^R, respectively. Because T^A and T^R may not always properly align their timestamps, they are further merged at line 6 by including their distinct timestamps and combining their common ones. The merged timestamps are returned for segmenting motion sensor data in the attacking phase.

Many researchers assumed the availability of key events and did not address the data segmentation challenge in the attacking phase; in other words, they only used the Identify-Keystroke-TimeWindows subroutine to perform motion sensor data segmentation in the training and attacking phases [1,3,7]. Cai and Chen used a library of keystroke motion waveform patterns to perform sensor data segmentation in the attacking phase [2]. However, this method requires a library to be pre-built; its accuracy depends on the quality of the library and the applicability of those patterns to different users.

The *Adjust-Keystroke-TimeWindows* subroutine adjusts the identified keystroke time windows in both training and attacking phases because some adjacent time windows may overlap and incur accuracy. For every two adjacent time windows W_j and W_{j+1}, the subroutine calculates the overlap between them at line 2. If they heavily overlap (i.e., the overlap region is greater than a certain percentage threshold, *overlap_threshold*, of the timespan between their corresponding key down events at line 5), the subroutine marks both of them as heavily overlapped time windows at line 6. If they slightly overlap, the subroutine adjusts their boundary to be the middle of the overlapped region at line 8. Finally all the heavily overlapped time windows are discarded at line 9, and the remaining time windows are returned at line 10. This adjustment step was not considered in any existing work; however, we observed in our experiments that about 5% of the identified time windows (either with or without using key events) heavily overlap (with *overlap_threshold* = 80%), and this adjustment can indeed improve the overall inference accuracy (Sect. 4.4) by approximately 1%.

3.3 Training Data Screening

Training data screening is one key technique that we designed to address the data quality challenge in cross-site input inference attacks. It calculates character-specific quality scores for individual keystrokes, and only uses the motion sensor data of good-quality keystrokes to train the classifier. In signal processing, the signal to noise ratio (SNR) is a commonly used quality estimation metric. Calculating SNR requires the characterization of the noise based on either the standard deviation of the random noise or the power spectrum density of the non-random noise. However, motion sensor data in input inference attacks may contain mixed random and non-random noises which are introduced from multiple sources such as human body movements. Therefore, there is no standard way to characterize the noises, and computing SNR in input inference attacks will not be reliable.

We propose a unique motion sensor data quality estimation algorithm *Estimate-Keystroke-Data-Quality* for screening the training data as shown in

Fig. 3. Overall, given m keystrokes of a specific user for a specific key, the algorithm first calculates their mean values of acceleration forces and rotation rates to obtain six averaged waveforms $\bar{c}$ for $c \in \{x, y, z, \alpha, \beta, \gamma\}$ at line 1; it then compares the waveforms of each individual keystroke with the averaged waveforms to calculate a quality score for the keystroke from line 3 to line 7. While it is not reliable to directly compute SNR, averaging m measurements of a signal can ideally improve the SNR in proportion to the $\sqrt{m}$ [4]. This is the reason why our algorithm uses the averaged waveforms as the reference to calculate quality scores. In more details, at line 4, the algorithm computes cross correlation values s_i^c between each individual keystroke K_i and the averaged waveforms $\bar{c}$ for each c to represent their level of similarity. Then at line 5, it computes weights w^c for each c by averaging the cross correlation values of m keystrokes. At line 6 and line 7, it computes a quality score Q_i for each keystroke K_i by adding its weighted cross correlation values on x, y, z, α, β, and γ.

Estimate-Keystroke-Data-Quality (K)
// $K = (K_1, K_2, \cdots, K_m)$: m keystrokes of a user for a specific key
// $K_i = ((x^i_{t_n}, y^i_{t_n}, z^i_{t_n}, \alpha^i_{t_n}, \beta^i_{t_n}, \gamma^i_{t_n}), (x^i_{t_{n+1}}, y^i_{t_{n+1}}, z^i_{t_{n+1}}, \alpha^i_{t_{n+1}}, \beta^i_{t_{n+1}}, \gamma^i_{t_{n+1}}), \cdots,$
$(x^i_{t_{n+j}}, y^i_{t_{n+j}}, z^i_{t_{n+j}}, \alpha^i_{t_{n+j}}, \beta^i_{t_{n+j}}, \gamma^i_{t_{n+j}})$: acceleration forces x, y, z
and rotation rates α, β, γ of the i-th keystroke from time t_n to t_{n+j}
// $Q = (Q_1, Q_2, \cdots, Q_m)$: quality scores for m keystrokes in K
1 calculate each $\bar{c} = (\bar{c}_{t_n}, \bar{c}_{t_{n+1}}, \cdots, \bar{c}_{t_{n+j}})$ for $c \in \{x, y, z, \alpha, \beta, \gamma\}$
where $\bar{c}_{t_k}$ =Mean $(c^1_{t_k}, c^2_{t_k}, \cdots, c^m_{t_k})$
2 $s = ()$ // Cross-correlation values of m keystrokes for $x, y, z, \alpha, \beta, \gamma$
$w = ()$ // Weights for $x, y, z, \alpha, \beta, \gamma$
3 **for each** K_i **in** $(K_1, K_2, \cdots, K_m)$
4 calculate each s_i^c = Cross-Correlation $((c^i_{t_n}, c^i_{t_{n+1}}, \cdots, c^i_{t_{n+j}}), \bar{c})$ for $c \in \{x, y, z, \alpha, \beta, \gamma\}$
5 calculate each w^c = Mean $(s_1^c, s_2^c, \cdots, s_m^c)$ for $c \in \{x, y, z, \alpha, \beta, \gamma\}$
6 **for each** K_i **in** $(K_1, K_2, \cdots, K_m)$
7 $Q_i = s_i^x \times w^x + s_i^y \times w^y + s_i^z \times w^z + s_i^\alpha \times w_i^\alpha + s_i^\beta \times w^\beta + s_i^\gamma \times w^\gamma$
8 **return** Q

Fig. 3. Keystroke data quality estimation algorithm

This algorithm does not rely on any special heuristic or threshold, and it can be executed online efficiently with polynomial time complexity. Using this algorithm, the training data screening component computes quality scores of individual keystrokes of a user for a specific key, and ranks the keystrokes based on their quality scores. Later, only a certain percent of top-quality keystrokes will be selected for further processing and for training a classifier.

3.4 Fine-Grained Data Filtering

Fine-grained data filtering is the other key technique that we designed to address the data quality challenge in cross-site input inference attacks. It selects frequency bands for data filtering at a fine granularity to reduce the noise in the motion sensor data. As shown in Fig. 1, this filtering technique is applied to the screened data in the training phase to identify the most effective filters, which are used to reduce the noise in both the training and attacking phases.

Frequency domain data filtering is a commonly used noise reduction technique. In the context of input inference attacks, researchers applied filters with fixed bands [2], used interpolation-based data smoothing methods [3], or used Discrete Fourier Transformation (DFT) and inverse DFT methods [1]. All these methods essentially discard high-frequency components and are equivalent to using certain fixed-band low-pass filters; however, it is not shown in these studies that a fixed-band low-pass filter is most appropriate and effective.

We propose a fine-grained data filtering technique, in which the frequency bands are selected with varying lengths instead of being fixed, for example, to a low-pass or high-pass band; meanwhile, different frequency bands are selected to effectively attack different users. Specifically, our technique divides the entire frequency band into multiple finer-granularity sub-bands, iterates all the consecutive concatenations of one or multiple sub-bands, and selects the concatenated band that performs the best as the frequency band for a particular user.

One typical band division method is the $\frac{1}{n}$ Octave method [6], which first divides an entire frequency band into two halves, then recursively divides the low frequency half multiple times in the same manner, and finally further equally divides each current sub-band into n new sub-bands. The $\frac{1}{n}$ Octave method favors low frequency components by dividing them into finer-granularity sub-bands, and it is often used in processing audio data that are dominated by low frequency components [6]. We use the $\frac{1}{2}$ Octave method to divide the entire frequency band (i.e., 0 Hz to 30 Hz, which is the mirrored first half of 60 Hz sampling frequency in Google Chrome used for collecting our motion sensor data) into ten sub-bands (four recursive divisions and one final $\frac{1}{2}$ division), but merge the first two low-frequency sub-bands into one due to their small sizes; the second column of Table 1 lists the nine final Octave sub-bands. Alternative division methods exist, for example, a straightforward method is to divide the entire frequency band into sub-bands with an equal size; we also use this method to derive nine equal sub-bands as shown in the third column of Table 1 as a comparison.

Table 1. Nine $\frac{1}{2}$ Octave and nine equal sub-bands

Sub-band index	1/2 Octave sub-bands (Hz)	Equally divided sub-bands (Hz)
1	0–1.88	0–3.33
2	1.88–2.65	3.33–6.67
3	2.65–3.75	6.67–10
4	3.75–5.3	10–13.33
5	5.3–7.5	13.33–16.67
6	7.5–10.61	16.67–20
7	10.61–15	20–23.33
8	15–21.21	23.33–26.67
9	21.21–30	26.67–30

From the nine sub-bands divided using either method, we further derive 45 consecutively concatenated bands from nine length-one concatenations, eight length-two concatenations, and finally to one length-nine concatenation. All these 90 bands together with a commonly used simple (less configuration effort) yet efficient Infinite Impulse Response filter [6] are applied individually and independently to our screened motion sensor data; later, the band for the best-performing classifier is selected as the most effective frequency band for a particular user, and will be used in the attacking phase.

3.5 Feature Extraction and Model Training

As shown in Table 2, we use 30 types of raw and derived motion sensor data of a given keystroke to extract statistical features. Sixteen types of data are singletons, and fourteen are pairs. The 16 singletons include acceleration forces (x, y, z), rotation rates (α, β, γ), the magnitude of acceleration forces (M^A), the magnitude of rotation rates (M^R), and all their first differences (D(x), D(y), D(z), D(α), D(β), D(γ), D(M^A), D(M^R)). The 14 pairs include three pairs of acceleration forces $((x, y), (y, z), (z, x))$, three pairs of rotation rates $((\alpha, \beta), (\beta, \gamma), (\gamma, \alpha))$, one pair of the magnitudes of acceleration forces and rotation rates $((M^A, M^R))$, and seven pairs of their corresponding first differences.

From the 16 singletons, the feature extraction component extracts (from both time and frequency domains) nine types of statistical features: maximum value, minimum value, mean value, variance, standard derivation, root mean square (RMS), skewness, kurtosis, and area under curve (AUC); as a result, $16 \times 2 \times 9 = 288$ features are extracted from the 16 singletons. Given the motion sensor data of a keystroke in the time domain, the maximum and minimum values are the peak and valley values; the mean value is the averaged amplitude; the variance, standard deviation, and RMS measure the deviations on amplitude; the skewness measures the symmetry of the motion sensor data; the kurtosis measures whether the motion sensor data are heavily or lightly tailed in comparison to a normal distribution; the AUC measures the power of the motion sensor data. In the frequency domain, all these nine features statistically measure the distribution of frequency components of the motion sensor data. From the 14 pairs, the component extracts their 14 cross correlation values in the time domain. Therefore, in total, $288 + 14 = 302$ statistical features are extracted from the motion sensor data of a keystroke, and are used in training and prediction.

In the model training, we experimented with a variety of machine learning algorithms using Weka [10], and observed that using the default Sequential Minimal Optimization (SMO) [5] for training a Support Vector Machine (SVM) classifier (with default parameters and the default linear kernel) outperforms all the other algorithms (with their default configurations) in inference accuracy. We only present the evaluation results of using SMO for SVM in the next section.

Table 2. Extracted statistical features

Data (16 singletons and 14 pairs)		Domain	Extracted features	Number of features
x	D(x)	Time & Frequency	Max, Min, Mean, Variance, Standard deviation, Root mean square, Skewness, Kurtosis, Area under curve	$2 \times 2 \times 9 = 36$
y	D(y)			$2 \times 2 \times 9 = 36$
z	D(z)			$2 \times 2 \times 9 = 36$
α	D(α)			$2 \times 2 \times 9 = 36$
β	D(β)			$2 \times 2 \times 9 = 36$
γ	D(γ)			$2 \times 2 \times 9 = 36$
M^A	D(M^A)			$2 \times 2 \times 9 = 36$
M^R	D(M^R)			$2 \times 2 \times 9 = 36$
(x, y)	(D(x), D(y))	Time	Cross correlation	$2 \times 1 \times 1 = 2$
(y, z)	(D(y), D(z))			$2 \times 1 \times 1 = 2$
(z, x)	(D(z), D(x))			$2 \times 1 \times 1 = 2$
(α, β)	(D(α), D(β))			$2 \times 1 \times 1 = 2$
(β, γ)	(D(β), D(γ))			$2 \times 1 \times 1 = 2$
(γ, α)	(D(γ), D(α))			$2 \times 1 \times 1 = 2$
(M^A, M^R)	(D(M^A), D(M^R))			$2 \times 1 \times 1 = 2$

*D() is the first differences of a sequence, e.g., $\mathrm{D}(x) = (x_2 - x_1, x_3 - x_2, \cdots, x_n - x_{n-1})$.

4 Evaluation

4.1 Data Collection

Participants: With the IRB approval from our university, we recruited 20 adults for data collection. We asked all the participants to use their own or our provided Android smartphones, and use the Google Chrome Web browser with the default Google Keyboard to perform input tasks. In the recruitment process, potential participants were administered the informed consent.

Websites Setup: We created two websites: one of them (i.e., the "malicious" website) uses JavaScript code to perform cross-site motion sensor data collection from the other website (i.e., the "victim" website). From the "victim" website that we own, we were also able to collect the key events for segmenting the motion sensor data, and the tapped characters for labeling the corresponding individual keystrokes. The "victim" website contains four webpages. Each webpage displays a different letter pangram and a different digit pangram, and asks our participants to type the two pangrams in two input fields, respectively. As shown in Table 3, each letter pangram is a sentence using every letter of the alphabet exactly once so that a participant does not need to type a longer sentence in each input field. Each digit pangram contains ten unique digits, and three special characters at the left, middle, and right parts of the keyboard.

Procedure and Dataset: We asked every participant to perform four tasks by visiting the four webpages and typing the displayed pangrams in each session. We asked each participant to complete a total number of 26 sessions in two weeks,

Table 3. Pangrams used in the study

Webpage	Letter pangrams	Digit pangrams
1	cwm fjord bank glyphs vext quiz	@83294&60571)
2	squdgy fez blank jimp crwth vox	&56920)71438@
3	tv quiz drag nymphs blew jfk cox	)45372&80916@
4	q kelt vug dwarf combs jynx phiz	@28513)97604&

but allowed them to do so at any places; therefore, we were able to collect a relatively large amount of data from participants in their real daily environments without any restriction. Overall, we collected $4 \times 26 = 104$ keystroke samples for each of the 39 characters (lower-case letters, digits, and three special characters) from each individual participant. Due to the error correction in typing, our participants indeed contributed 17,571 additional keystroke samples in their sessions. As a result, the total number of keystroke samples in our final dataset is $104 \times 39 \times 20 + 17,571 = 98,691$.

4.2 Accuracy Metrics and Evaluation Methodology

To evaluate the accuracy of a trained multi-class classifier, we first count the *true positive* (TP), *false positive* (FP), *true negative* (TN), and *false negative* (FN) numbers. For a given class (e.g., letter "a"), a true positive is an instance correctly predicted as belonging to that class (e.g., letter "a" is correctly predicted as "a"), a false positive is an instance incorrectly predicted as belonging to that class (e.g., letter "b" is incorrectly predicted as "a"), a true negative is an instance correctly predicted as not belonging to that class (e.g., letter "b" is correctly predicted not as "a"), a false negative is an instance incorrectly predicted as not belonging to that class (e.g., letter "a" is incorrectly predicted not as "a"). We further calculate *false positive rate* (FPR), *precision*, *recall* (i.e., true positive rate, or TPR), and *F-measure* accuracy metrics for each class, and average their corresponding values across classes as the accuracy for the multi-class classifier. The F-measure metric is the harmonic mean of precision and recall; thus, we mainly present and analyze the results based on this metric.

In the evaluation, our classifier is trained and assessed using the 10-fold cross validation, and we run the cross validation for 5 rounds and present their averaged results. We evaluate the inference accuracy explicitly on all the three charsets: the letter charset (i.e., 26 lower-case letters), the digit charset (i.e., 10 digits together with 3 special characters), and the mixed charset (i.e., all the 39 characters). This is because in real scenarios, an attacker may know the type information of an input regarding if it is a letter or digit, and can directly use a classifier specific to the inference of either letters or digits.

4.3 Overall Accuracy with Training Data Screening

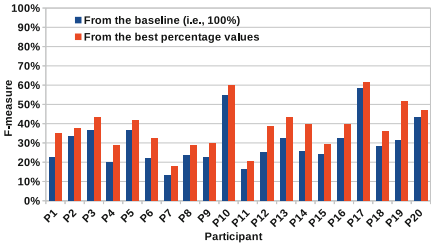
We evaluate the overall accuracy of our inference attacks with the focus on quantifying the extent to which our training data screening technique can improve the accuracy. We use the keystroke data quality estimation algorithm (Fig. 3) to rank the keystrokes of a given participant for each specific key, and select a certain percent of top-quality keystrokes for training a classifier and performing the 10-fold cross validation. Specifically, we choose 10 percentage values from 0.1 (i.e., 10%), 0.2 (i.e., 20%),..., to 1.0 (i.e., 100%). In particular, the 100% value means that all the keystrokes will be used in training, and the corresponding inference accuracy serves as the baseline in our accuracy comparison. Given a specific percentage value and a specific charset, we ensure that the sample sizes are roughly equal for different characters to avoid training a classifier using unbalanced data. Eventually, the percentage value that yields the highest inference accuracy will be selected for each participant as the best percentage value for screening the training data. In this percentage value selection process, fine-grained data filtering is turned off to avoid circular dependency.

Figures 4(a), (b), and (c) illustrate the overall inference accuracy for the 20 participants on the three charsets, respectively. In each subfigure, we compare the inference accuracy (i.e., F-measure) for each participant between that from the baseline (i.e., 100%) and that from his or her best percentage value. Regarding the inference accuracy from the baseline, the F-measure scores for the 20 participants vary from 12.97% to 58.14% with the average at 30.12% for the letter charset, from 21.21% to 66.91% with the average at 39.71% for the digit charset, and from 9.17% to 46.97% with the average at 23.45% for the mixed charset. By using training data screening with the best percentage values, the F-measure scores for the 20 participants are improved (upon those of the baseline) from 3.41% to 20.45% with the average at 8.03% for the letter charset, from 1.96% to 18.75% with the average at 9.93% for the digit charset, and from 2.8% to 16.96% with the average at 7.21% for the mixed charset. The inference accuracy is improved for all the 20 participants, demonstrating that our training data screening technique is indeed effective.

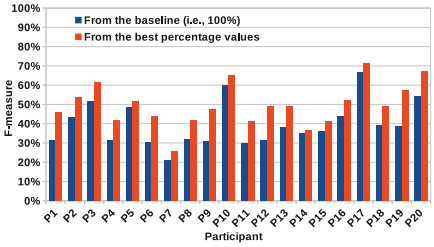
Two additional observations from Fig. 4 are worth mentioning. One is that for almost all the participants, the corresponding inference accuracy on the digit charset is higher than that on the letter charset, which is further higher than that on the mixed charset. For example, for participant P12, the inference accuracy on the digit, letter, and mixed charsets is 49.13%, 38.63%, and 31.29%, respectively. The other observation is that the relative inference accuracy differences among the participants are highly consistent across the three charsets. For example, the inference accuracy for participant P7 is the lowest among all the participants across the three charsets, while that for participant P17 is always the highest.

4.4 Overall Accuracy with Fine-Grained Data Filtering

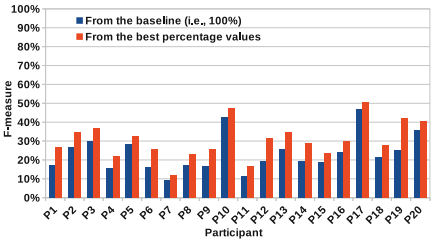
Our fine-grained data filtering technique (Sect. 3.4) improves the inference accuracy for the majority of the participants; meanwhile, the $\frac{1}{2}$ Octave method performs better on the digit charset, while equally dividing the entire frequency band



(a) Letter charset



(b) Digit charset



(c) Mixed charset

Fig. 4. Overall accuracy on letter, digit, and mixed charsets

performs better on the mixed charset. With this further improvement, our input inference attacks overall (1) achieve 2.45%, 39.74%, 38.77%, and 38.83% regarding FPR, precision, recall (TPR), and F-measure, respectively, on the letter charset, (2) achieve 4.1%, 51.45%, 50.75%, and 50.79% regarding the four metrics, respectively, on the digit charset, and (3) achieve 1.81%, 32.04%, 31.42%, and 31.36% regarding the four metrics, respectively, on the mixed charset. Note that *a smaller training dataset can still achieve good inference accuracy*. For example, using 41 keystroke samples for each character in 10-fold cross validation (thus less than 37 samples for training) can still give us a 33% F-measure score for the letter charset.

4.5 Further Overall Accuracy Comparison and Analysis

Because our trained classifier (using SMO for SVM) is a probabilistic classifier that predicts the probabilities over a set of classes, we further consider the top-n predicted results and define the *hit probability* as the probability that the ground truth is among them. This hit probability corresponds to the probability of hitting the ground truth in at most n tries of the top-n results. Figure 5 illustrates the hit probability curves from one try to four tries, for our input inference attacks denoted by the solid lines and for the random guessing attacks denoted by the dashed lines. The hit probability increases with the increase of the number of tries. For example, it increases from 41.5% in one try to 79.52% in four tries for our input inference attacks on the letter charset. Note that these

numbers are averaged over all the predictions across the participants. Our input inference attacks are much more effective than the random guessing attacks. For example, on the letter charset, our attacks are about 10.8 times and 5.2 times more effective than the random guessing attacks (i.e., guessing a letter from 26 possibilities) in one try and four tries, respectively.

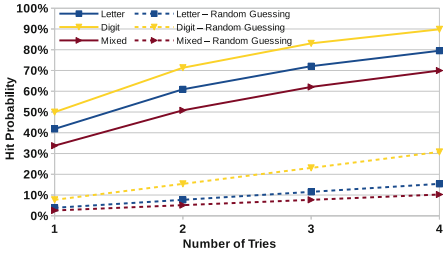


Fig. 5. Hit probability in one to four tries for three charsets

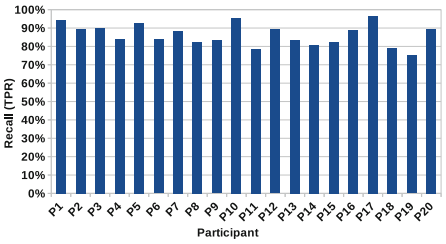


Fig. 6. Overall data segmentation accuracy

4.6 Accuracy of Sensor Data Segmentation Without Key Events

In this subsection, we evaluate the accuracy of the *Detect-KeyDown-Timestamps* subroutine by comparing its detection results with the collected ground-truth key down timestamps. This accuracy determines the accuracy of the *Segment-SensorData-Without-KeyEvents* algorithm shown in Fig. 2.

For the purpose of this evaluation, we need to define a new set of accuracy metrics. If a time window (identified by the *Identify-Keystroke-TimeWindows* subroutine in Fig. 2) for a detected key down timestamp contains any ground-truth key down timestamp, a true positive (TP) is counted; otherwise, a false positive (FP) is counted. If a ground-truth key down timestamp is not in any of those identified time windows, a false negative (FN) is counted. However, we are not able to count true negatives because they are simply not definable.

Because Google Chrome on Android does not report the key down and up events of special keys (e.g., caps lock key, keyboard switching key, and enter key) to the JavaScript code on regular webpages, we do not have the ground-truth to exclude the keystrokes for special keys, and our false positive numbers are unavoidably over-counted in this evaluation. Therefore, to represent the accuracy of the key down timestamp detection, it is more reasonable for us to use the recall (TPR) scores instead of the precision or F-measure scores (which are affected by the over-counted false positives).

Figure 6 illustrates that the recall scores are above 80% for the majority of the participants, demonstrating that our *Segment-SensorData-Without-KeyEvents* algorithm is indeed effective in segmenting sensor data for true keystrokes. In real attacks without key events, the overall input inference accuracy depends on the data segmentation accuracy, and thus could be slightly reduced.

5 Conclusion

We investigated severe cross-site input inference attacks that may compromise the security of every mobile Web user, and quantified the extent to which they can be effective. We formulated our attacks as a typical multi-class classification problem, and built an inference framework that trains a classifier in the training phase and predicts a user's new inputs in the attacking phase. We addressed the data quality and data segmentation challenges in our attacks by designing and experimenting with three unique techniques: training data screening, fine-grained data filtering, and key down timestamp detection and adjustment. We intensively evaluated our attacks and found they are effective. Our results demonstrate that researchers, smartphone vendors, and app developers should pay serious attention to the severe cross-site input inference attacks that can be pervasively performed, and should start to design and deploy effective defense techniques.

Acknowledgment. This research was supported in part by the NSF grant DGE-1619841.

References

1. Aviv, A.J., Sapp, B., Blaze, M., Smith, J.M.: Practicality of accelerometer side channels on smartphones. In: Proceedings of the Annual Computer Security Applications Conference (ACSAC), pp. 41–50 (2012)
2. Cai, L., Chen, H.: On the practicality of motion based keystroke inference attack. In: Katzenbeisser, S., Weippl, E., Camp, L.J., Volkamer, M., Reiter, M., Zhang, X. (eds.) Trust 2012. LNCS, vol. 7344, pp. 273–290. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30921-2_16
3. Miluzzo, E., Varshavsky, A., Balakrishnan, S., Choudhury, R.R.: TapPrints: your finger taps have fingerprints. In: Proceedings of the International Conference on Mobile Systems, Applications, and Services, pp. 323–336 (2012)
4. Orfanidis, S.J.: Introduction to Signal Processing. Prentice-Hall Inc., Englewood Cliffs (1995)
5. Platt, J.: Sequential minimal optimization: a fast algorithm for training support vector machines. Technical report (1998)
6. Smith, S.W.: The scientist and engineer's guide to digital signal processing (1997)
7. Xu, Z., Bai, K., Zhu, S.: TapLogger: inferring user inputs on smartphone touchscreens using on-board motion sensors. In: Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 113–124 (2012)
8. Yue, C.: Sensor-based mobile web fingerprinting and cross-site input inference attacks. In: Proceedings of the IEEE Workshop on Mobile Security Technologies (2016)
9. Same Origin Policy. https://www.w3.org/Security/wiki/Same_Origin_Policy
10. Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ml/weka/