iStelan: Disclosing Sensitive User Information by Mobile Magnetometer from Finger Touches

Reham Mohamed, Habiba Farrukh, Yidong Lu, He Wang, and Z. Berkay Celik
{raburas,hfarrukh,lu805,hw,zcelik}@purdue.edu
Purdue University

ABSTRACT

We show a new type of side-channel leakage in which the built-in magnetometer sensor in Apple's mobile devices captures the touch events of users. When a conductive material such as the human body touches the mobile device screen, the electric current passes through the screen capacitors generating an electromagnetic field around the touch point. This electromagnetic field leads to a sharp fluctuation in the magnetometer signals when a touch occurs, both when the mobile device is stationary and held in hand naturally. These signals can be accessed by mobile applications running in the background without requiring any permissions. We develop iStelan, a three-stage attack, which exploits this side-channel to infer users' applications and touch data. iStelan translates the magnetometer signals to a binary sequence to reveal users' touch events, exploits touch event patterns to fingerprint the type of application a user is using, and models touch events to identify users' touch event types performed on different applications. We demonstrate the iStelan attack on 22 users while using 7 popular app types and show that it achieves an average accuracy of 90% for disclosing touch events, 74% for classifying application type used, and 73% for detecting touch event types.

KEYWORDS

Side-channel attacks, user privacy, information leakage

1 INTRODUCTION

High-resolution sensors present in modern-day mobile and IoT devices have enabled diverse apps–from social networking to banking and healthcare–to be more autonomous, adaptive, and efficient. However, the data from mobile sensors enable adversaries to gain unauthorized access to sensitive user information, such as their habits, behaviors, and preferences [1], as well as physical attacks, such as [9, 39]. To prevent surreptitious access to mobile sensors, mobile operating systems such as Android and iOS have introduced sensitive and normal permission types according to the sensitivity of sensors accessed by an app [54]. These permissions require access requests granted by users to acquire privacy-sensitive sensors (e.g., cameras and GPS). The accelerometer, gyroscope, and magnetometer sensors, however, are generally not regulated by user-level and system-level access controls due to usability concerns and their impact on the app functionality, making them a prime target for information stealing attacks.

Proceedings on Privacy Enhancing Technologies 2023(2), 79–96

https://doi.org/10.56553/popets-2023-0042

Previous efforts have explored side-channel privacy risks through sound [36] and light [53] to infer the touch events of users on smartphone screens. Sound signals were also used to infer the target user's handwriting [63]. These attacks, however, require explicit user permission that invalidates the adversary's prior knowledge about the system and require an attacker to obtain the signals by external measurement equipment within physical proximity during the attack. To address these problems, recent works have shown that the accelerometer and gyroscope on smartphones can be used to infer private information such as a user's touch behavior [29, 34, 35, 38, 62]. However, these attacks depend on significant movement in the phone body while the user touches the screen and are contaminated with the environmental factors that cause changes in motion. Consequently, they require users to hold the phone in their hands and type with specific fingers. Thus, they are ineffective when the phone is stationary or encounters a small motion, for instance, when the phone is placed on a table or a user holds the phone temporarily stationary.

In this paper, we present a new side-channel leakage in iOS devices, including iPhones and iPads, which can be used to infer users' touch data from the onboard magnetometer. We observe that human touch on the device screen generates an electromagnetic field around the point of touch. This electromagnetic impact is generated from capacitive touchscreens that consist of conductive transparent layers below their glass sheet. Particularly, a capacitance change occurs in the electrodes of the conductive layer when the human finger touches the screen due to the conductive nature of the human body [49]. This change leads to a displacement current producing an electromagnetic field. Due to the magnetometer's high resolution and close placement relative to the screen, a noticeable impact is created on the magnetometer signal.

We tested our observation on ten mobile devices from different smartphone vendors. We found that this electromagnetic impact of user's touch is prominent on Apple's iPhone and iPad (Section 2), which hold 20.8% and 64.6% of the smartphones and tablets market share, respectively [55]. The side-channel information leaked by the magnetometer eliminates the limitations of previous work using accelerometer and gyroscope, which require a broad range of motion in the phone to detect touch events.

We introduce iStelan, which takes a new approach to analyzing leaked magnetometer signals to infer users' touch and application information. iStelan stealthily collects magnetometer data from the victim's phone in the background without access to other sensitive information. The collected data is then used in three attack stages. First, we introduce a binary touch extractor extending a Deep Neural Network (DNN) classifier with a sequence-to-sequence mapping to uncover binary touch events from 3D magnetometer signals, which are only accessible to foreground apps in practice.

From this, we build an app classifier to identify the type of app a user is using from fingerprints of different apps learned through a Long Short-term Memory Network (LSTM). Lastly, the touch event type detector integrates the binary touch sequence of each app type into a Hidden Markov Model (HMM) to label users' touch event sequence with touch actions, such as swiping and typing.

Unlike previous works, we do not directly infer touch locations. However, the private user data inferred by iStelan, including app fingerprinting and touch events information enables an adversary to perform UI state hijacking attacks (e.g., confusion attacks [3]), where an adversary shows an interface that mimics the app the user is using. This allows an adversary to steal users' passwords or personal data and access sensitive information, such as camera images [11]. Additionally, starting with iOS 14.5, Apple's App Tracking Transparency (ATT) framework requires apps to explicitly request user permission to track users' activity across various apps and to access their advertising IDs. An adversary using iStelan attack could evade this requirement by stealthily inferring users' most popular apps in the background. This allows the adversary to display targeted advertisements without gaining explicit app tracking permission. Furthermore, a malicious app using iStelan attack could act as a data broker to sell users' data to interested third parties. For example, knowing that a user is using a travel app, ticket booking platforms could use this data to adjust their prices. Similarly, a user using a health app could be the target of medical insurance companies to provide her with tailored offers.

We evaluate iStelan attack by collecting a data set from 22 users. We perform experiments with 7 different mobile app types from popular categories on the app store. iStelan discloses users' touch events with an average 90% accuracy, correctly classifies the app that a user is using with an average 74% accuracy, and detects the touch event labels of users with an average 73% accuracy. We additionally perform comprehensive experiments to demonstrate the effectiveness and performance of iStelan on unseen apps and under various environmental conditions.

In summary, we make the following contributions:

- We show a new privacy leakage through the magnetometer on iOS devices (iPhone and iPad) due to the electromagnetic impact of human touch on capacitive screens.
- We introduce iStelan, which exploits 3D magnetometer signals to stealthily extract users' touch actions in the background and develop machine learning and statistical models for app fingerprinting and inferring touch events.
- We evaluate iStelan attack on a data set collected from 22 users and demonstrate its effectiveness in exposing users' binary touches, app types, and touch event types.

2 ISTELAN INSIGHTS

2.1 Structure of Touch Screens

Capacitive touch screens have gained huge popularity in the smartphone industry due to their better optical performance and multi-touch support [50]. Touch screens use electrodes to sense the conductive properties of a user's finger. Figure 1 shows the basic structure of a capacitive smartphone touch screen. It consists of two diamond electrode grids made of conductive transparent material

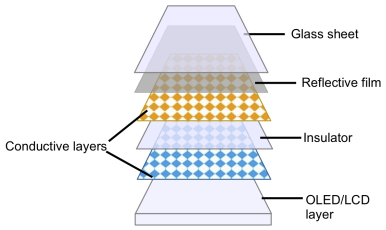


Figure 1: Structure of a capacitive smartphone touch screen.

with an optically clear insulator in the middle. These conductive grids are layered below a sheet of glass or film. When a voltage is applied to the electrode grids, electric fields are projected on the surface of the glass sheet [47]. Since the human body is a conductive material, an electric change is detected when a finger approaches the touch electrodes [49]. The processor then registers a touch and determines its location.

The organization of electrodes in the form of a matrix of rows and columns allows localization of the touch event at the coordinates where the electric change occurs. There are two variants of capacitive touch technology: self-capacitance and mutual capacitance. In self-capacitance, the capacitance occurs between the conductive material (human finger) and electrode, while in mutual capacitance, it occurs between two adjacent electrodes. Most modern smartphones rely on mutual capacitance that supports more advanced touch gestures.

2.2 Effect of Touch Events on Magnetometer

Capacitive touch screens used in mobile devices comprise several layers with a conductive material under the glass sheet containing the electrode matrix. As the user's finger touches the screen, a capacitance change occurs at the point of touch resulting in a change in the electromagnetic field around this point. Based on Maxwell theory [32], the electric field between capacitors sets up a displacement current $\mathtt{I(t)}$. This displacement current causes a magnetic field to be generated inside and outside capacitor plates [33]. The magnetic field $\mathtt{B(a, t)}$ at a point at distance $\mathtt{a}$ from the capacitor is computed using Ampere's law: $\mathtt{B(a, t)} \propto \mathtt{I(t)}/2\pi\mathtt{a}$.

We observed that this change in the magnetic field could be detected via the magnetometer embedded in smartphones since it is placed at millimeters distance from the capacitive screen and has a high resolution (up to $\mathtt{0.15}\ \mu\mathtt{T}$). Though this magnetic change is relatively small compared to the ambient noise in the magnetometer readings, we show that this effect can be exploited to leak touch events through background apps. Furthermore, this data leakage can be analyzed to understand the touch behavior of users to reveal their private information, such as their app usage patterns.

Locating the Leakage in Mobile Devices. Many smartphone vendors have adopted capacitive touchscreen technology. To investigate the existence of the described electromagnetic impact of user's touch on different smartphones, we tested various versions of smartphones from different vendors, including Apple, Google, Samsung, and Huawei. For each device, we installed an app that records the magnetometer and touch data for 4 minutes on average. We then compare the touch data (ground truth) with the magnetometer readings to spot the touch effect.

Table 1: Impact of touch events on magnetometer in different phones and tablets.

Phone version	Type of screen	Magnetometer	Magnetometer effect
Google Pixel XL	Capacitive/OLED	AKM AK09915	✗
Google Pixel 3a	Capacitive/OLED	STMicro LIS2MDL	✗
Huawei P20	Capacitive/LCD	AKM	✗
Samsung S10	Capacitive/OLED	AKM AK09918C	✗
iPhone 6	Capacitive/LCD	AKM AK8963	✓
iPhone 8	Capacitive/LCD	Alps e-Compass	✓
iPhone XS	Capacitive/OLED	Unknown	✓
iPhone 12	Capacitive/OLED	Unknown	✓
iPad (6^{th} generation)	Capacitive/LCD	Unknown	✓
iPad Air (4^{th} generation)	Capacitive/LED	Unknown	✓

Table 1 summarizes the type of screen and the existence of magnetic effect on these smartphones. We observed that the effect on the magnetometer due to users' touch is more significant on Apple devices. The effect is consistently observed on different iPhone and iPad versions. However, the electromagnetic impact of a user's touch is not significantly visible on other mobile devices. Our experiments show that the main reason for this difference is that, unlike other smartphones, iPhone devices use an adhesive material between the screen layers which removes any air gaps and reduces the thickness of the display. This allows small magnetic fields from the touch screen to be sensed using the underlying magnetometer.

We additionally observed that the internal structures of different phones are different. For example, in Apple devices, the main board, which contains the sensors, is placed directly below the touchscreen [21] while Samsung devices include a mid-frame between the screen and the main board [22]. This could also mitigate the touch screen effect on the magnetometer, which makes it indistinguishable from the sensor noise. Therefore, we propose the iStelan side-channel affecting iOS devices used by over a billion users worldwide [55].

Demonstrating the Leakage Signal. Figure 2 shows the effect of touch on the three axes of the magnetometer data collected with a sampling rate of 100Hz from an iPhone XS after applying a moving-average smoothing filter for noise removal. During a touch event, the magnetic field changes cause a fluctuation in the signal in the three axes of the magnetometer. In some cases, these fluctuations are small compared to the ambient noise in the signal. Yet, by combining the 3-D magnetometer data, we show that iStelan is able to detect the touch events with 90% average accuracy. We obtained similar results on other tested iPhone and iPad devices.

Currently, iOS supports a sampling rate of up to 100Hz for collecting magnetometer data. However, we also study the impact of touch on magnetometer data collected at lower sampling rates, ranging from 10Hz to 100Hz. We observe that the magnetic effect due to users' touches is not affected by the sampling rate of the magnetometer signal (Detailed in Section 5). Therefore, any future iOS update limiting the sampling rate for the magnetometer sensor would not impact the success of iStelan attack.

Additionally, we compare the effect of touch on the magnetometer with other motion sensors in the phone, i.e., accelerometer and gyroscope. These sensors were previously exploited as side channels for leaking users' touch events [35, 62]. However, attacks based on these sensors require a significant phone movement to detect a touch event. When the phone is stationary or encounters a small motion, these attacks fail since there is no effect on the accelerometer/gyroscope. In contrast, the impact of touch on the magnetometer data does not rely on any movement. Figure 3 shows the impact

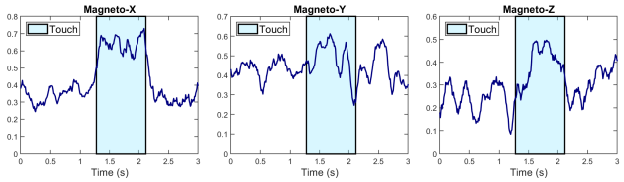


Figure 2: Effect of touch event on 3 axes of the magnetometer. The blue area shows the occurrence of a touch event.

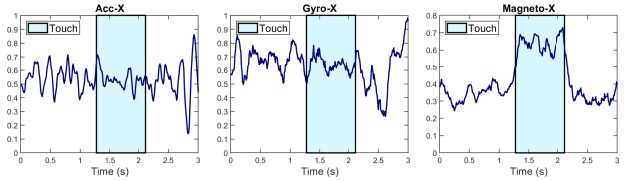


Figure 3: Effect of touch events on different motion sensors. The blue area shows the occurrence of a touch event.

Table 2: Comparison of data leakage constraints between iStelan and prior work.

Approach	(1)†	(2)‡	Restrictions
Motion Sensor Attacks			
AlphaLogger [23]	✓	✗	Hold phone with both hands and press keys with thumb
FreqKey [52]	✓	✗	Hold phone in right hand and press keys with thumb
TouchLogger [7]	✓	✗	Hold phone in hand
TapLogger [62]	✓	✗	Hold phone in left hand and tap with the right hand forefinger
ACCessory [38]	✓	✗	Hold phone in the landscape orientation, type with both thumbs
TouchSignatures [34]	✓	✗	Hold phone in one hand and use the same hand's thumb Use both hands to perform the touch
Magnetometer-based Attacks			
MagneticSpy [31]	✗	✓	Does not evaluate their system on users
DeepMag [37]	✗	✓	Does not evaluate their system on users
MagSnoop [13]	✗	✓	Does not evaluate their system on users
MagTheif [40]	✓	✓	None
iStelan	✓	✓	None

† (1) The phone is in motion, and the user is stationary. ‡ (2) The phone is stationary

of a touch event on the accelerometer, gyroscope, and magnetometer data of a phone placed on a table. All signals are shown after pre-processing for noise removal[1]. The accelerometer/gyroscope signals are not affected due to the lack of significant movements in the phone body. In contrast, the magnetometer shows a sharp signal fluctuation when the touch event occurs.

2.3 Comparison with Prior Work

Table 2 compares iStelan with previous works that proposed side-channel attacks using motion sensors to infer sensitive touch data. Although these works allow for stronger attacks than iStelan since they infer sensitive information such as keystrokes and passwords, they rely on the effect of hand motion on the motion sensors while typing. Therefore, they require the user to be stationary (sitting or standing) while holding the phone in hand. Furthermore, these systems add predefined restrictions on how the user must use the phone to be able to perform the side-channel attack. For example, AlphaLogger [23], FreqKey [52], and ACCessory [38] require the user to hold the phone in hand and type using the thumb. This ensures that the phone swings more violently while the user is typing, providing a more significant effect on motion sensors. Additionally, ACCessory requires the phone to be in landscape orientation to guarantee a wider range of motion.

[1]We obtained similar results for other axes of three sensors, as shown in Appendix C.

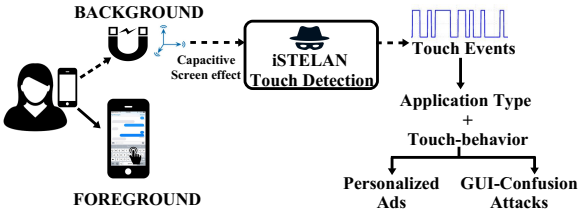


Figure 4: iStelan's threat model. The dotted lines indicate stealthy data collection from the magnetometer and extraction of touch events in the background. The solid lines illustrate the app and touch event type inference of the attacker.

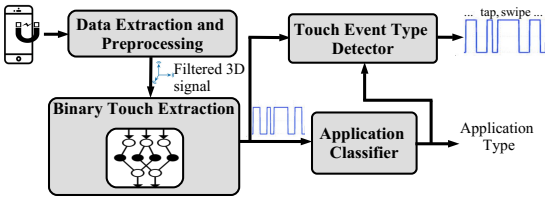


Figure 5: Overview of iStelan architecture

While these approaches achieve high accuracy, these additional restrictions make their attack highly ineffective in practical scenarios where the user holds the phone in hand in any orientation and uses it freely. Moreover, none of the systems could infer touches when the phone is stationary, e.g., placed on a table, such as in the case of iPad, fixed on a tripod, or even held more stiffly in hand. In contrast, iStelan infers touch activity and app types in more practical scenarios since it does not impose any restrictions on how the user holds the phone. Therefore, iStelan's attack works both when the device is stationary or held naturally in hand.

We highlight that most of the previous attacks are ineffective when the user moves since the effect of touch on motion sensors is masked by the user's motion. This was also shown in FreqKey [52], where their keystroke inference attack accuracy decreases when the user is in motion. Similarly, we evaluated iStelan when the user is walking and found that the attack accuracy decreases due to the interference of motion with the touch pattern and the change in the ambient magnetic field (Section 5). However, the attack works for a wide range of scenarios when the user is sitting or lying.

Other works proposed magnetic-based side-channel attacks, such as MagneticSpy [31] and DeepMag [37] that infer the application type using fingerprinting of processor electromagnetic emission and splash screen color effect on LED. MagSnoop [13] also exploits the magnetometer sensor to infer payment tokens in magnetic secure transmission. These works evaluated their systems with different apps in an automated way without performing any experiments with users; therefore, they assume the phone is completely stationary. MagTheif [40] also used the EM effect on magnetometer to fingerprint apps and in-app services and proposed to use other motion sensors to filter human motion from the magnetometer signal. However, it is highly prone to variations due to background activity noises and ambient magnetic noises. We expand on iStelan's comparison with prior works in Section 7.

Responsible Disclosure. We contacted Apple's product security team to disclose our findings on the characteristics and steps of side-channel observed in Apple devices. The security team acknowledged our findings and will investigate this attack further.

2.4 Threat Model

We consider an adversary that controls a malicious native app installed on the victim's device, which accesses only the onboard magnetometer of the victim's mobile device while running in the background (See Figure 4). The app does not have access to other data, such as touch events, apps used or installed on the device, network traffic, and process statistics. The data collected by the app is stored locally on the phone or transferred to a remote server. Both methods incur minimal energy and computation overhead to remain undetected by users and the operating system (Section 5).

We show that an adversary is able to disclose the victim's touch events solely using the magnetometer signals. From this, the adversary uses pre-trained app fingerprints to identify the app being used by the victim and the victim's touch event types on the app. To achieve these, the adversary trains the models using the data collected from a set of users (17 users yields reasonable accuracy (Section 5)) and packages with the malicious app.

Using the victim's app type and touch event type, an adversary can invade users' privacy by identifying the victim's interests and browsing behavior. We show two case studies, GUI-confusion attacks (Appendix E.1) and targeted advertisement attacks (Appendix E.2), to demonstrate the practical attacks with iStelan side-channel. Furthermore, an adversary can use the inferred touch event types to predict users' behavior and activities on different apps [34]. For instance, if a user is using a shopping app, depending on the pattern of touch event types, an attacker could detect the frequency of her purchases. Previous works also showed that touch event types could be used for different purposes, including emotion detection by analyzing 'typing' and 'swiping' gestures [17] and stress level assessment based on smartphone gestures analysis, such as 'tap', 'scroll', 'swipe' and 'text writing' [14]. An adversary can exploit such analysis to infer more private information about users' moods, assess their interest in apps, and target them with different ads depending on their emotions and stress levels.

We note that all native iOS apps, by default, have access to motion sensors, and only web apps from iOS 12.2 and onwards require explicit user permission. Similarly, starting from iOS 5, an app can stay active in the background if it performs certain tasks (e.g., playing audio, accessing location, or using voice over IP). Thus, an adversary can easily mimic a benign legitimate app, such as a fitness tracking or music app, to stay active in the background and collect motion sensor data without requiring explicit permissions.

3 APPROACH OVERVIEW

Figure 5 presents the architecture of iStelan system. iStelan first splits the continuous 3D magnetometer sensor signals collected by a malicious app into given time intervals. Low-pass and moving average filters are then applied to each interval in each dimension to remove noise and smooth the signal. Thereafter, iStelan trains a DNN model with sequence-to-sequence mapping to transform the continuous 3D magnetometer signals into binary touch data. The touch events are represented as a binary sequence where 1 represents a touch, and 0 represents no touch.

From this, iStelan uses the touch events to learn the user's behavior. First, it fingerprints the app type that a user is using by exploiting the fact that different mobile apps exhibit different touch patterns based on their functionality. To achieve this, it builds a one-vs-all classifier to detect the app type. The extracted binary sequence is encoded into a feature vector and fed into bidirectional LSTM models. iStelan builds an LSTM model for every app type to detect the app patterns and uses the aggregation max function to output the app type.

In the last stage, the touch event type detector uses both touch data and the identified app type to obtain a sequence of labeled touch event types. We observe that the extracted binary touch sequence is conditionally dependent on a hidden sequence of touch types, which could be modeled as a Markov process. Additionally, the transition between touch types depends on user actions unique to each app behavior. Based on these observations, iStelan builds an HMM model for each application type, where the binary touch sequence is used to represent the observations and the touch-type labels for hidden states. The HMM model outputs a labeled touch sequence for a specific app type. These labels enable an attacker to infer what a user is doing on an app, such as whether the user swipes, taps, types, or long presses.

4 iStelan DESIGN

Implementing iStelan requires addressing several system challenges. (1) Magnetometer data extraction and preprocessing to filter signal noise (Section 4.1). (2) Translating continuous 3D magnetometer signal into a binary touch-event sequence (Section 4.2). (3) Building an app classifier to precisely assign touch events to an app type (Section 4.3). (4) Building an HMM model to infer the touch-type sequence of the user's touch events (Section 4.4).

4.1 Sensor Data Extraction and Preprocessing

The malicious app installed on the victim's phone collects the 3D magnetometer signals at a sampling rate of 100Hz. The collected data can be stored and processed locally or sent to a remote server in batches for analysis.

We segment and preprocess 3D magnetometer signals before extracting touch events. To detail, each dimension of the signal is first segmented into samples of non-overlapping intervals of window size (w_s). We set w_s=60 seconds based on our experiments as detailed in Section 5. Each sample is then normalized to the [0, 1] range and filtered to smooth the data and remove noise. To mitigate the effect of random samples and fluctuations, we apply a moving average filter with a smoothing factor of 0.1 secs.

The magnetometer signals may also yield high-frequency interference noise due to the residual magnetic distortions from magnetic parts placed on the phone, power supply current, and nearby ferromagnetic objects. To eliminate this interference, we apply an additional low-pass 6^{th} order Butterworth filter [48]. The Butterworth filter uses a maximally flat magnitude, which results in a flat frequency response in the passband below a cutoff frequency (ω_c). We observe that normal phone usage is unlikely to exceed 5 touches per second in our experiments. Thus, we set ω_c to 5.

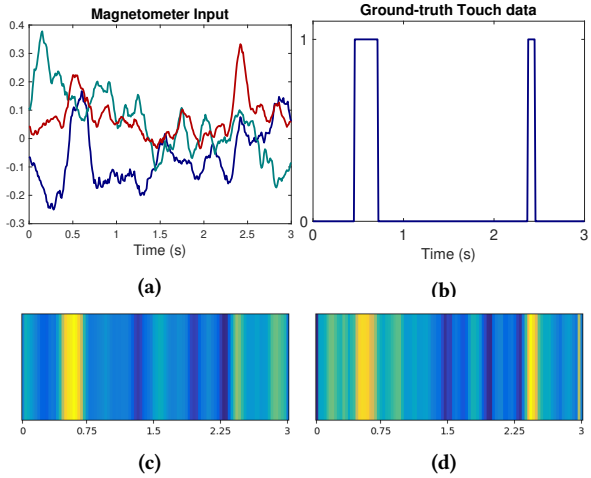


Figure 6: CNN feature extraction from magnetometer data. (a) 3D magnetometer signals, (b) ground truth touch data, and (c)-(d) different CNN filter outputs. The yellow regions represent the touch events shown in (b).

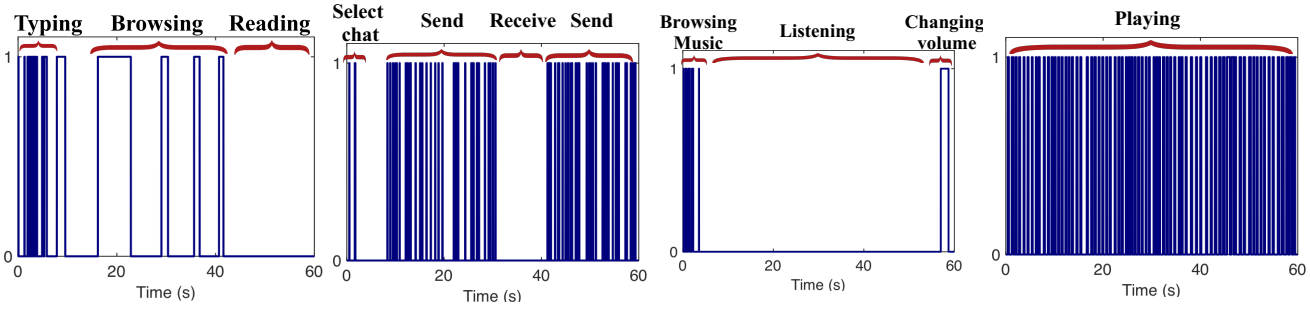
4.2 Binary Touch Extraction

The preprocessed magnetometer signals are fed into the binary touch extraction module. This module exploits the capacitive touch screen effect on the magnetometer to transform the continuous 3D magnetometer signal into touch events. Specifically, we represent the touch events for a given time interval as a binary sequence, where 1 represents a touch and 0 represents a no-touch.

To extract the touch events, we build a CNN-LSTM network that maps the continuous 3D magnetometer signal to a binary sequence of the same length. Lastly, a dense layer is used to classify each bit to 0 or 1. This combined learning process is effective because CNN layers exploit the spatial correlation, and LSTM layers capture the temporal dependencies in the magnetometer data.

Feature Extraction. To extract features from the magnetometer data, we use convolution neural network layers. The noise in the magnetometer is often variable due to different users and ambient environments. Due to this variability, it is infeasible to select one filter size for extracting touch event features that generalize for all conditions. A large filter size removes more noise, but it is prone to losing useful touch features, while a small filter retains features but is more affected by noise. To accommodate such variation, we use two parallel CNN layers with different filter sizes, which provide different views of the features extracted from the magnetometer data. The features extracted from both layers are concatenated and sent to the following layers for sequence mapping. In this way, we increase the likelihood of detecting useful features and decrease the negative effect of noisy samples.

We trained 15 different filters in each layer and collected a development dataset to select the optimal filter sizes and other hyperparameters. Figure 6 illustrates an example of features extracted through this process. The goal of iStelan is to transform the 3D input signal in Figure 6a to the binary touch sequence in Figure 6b. Figure 6c and Figure 6d show the heatmaps of two different CNN filters used for extracting these touch features. The yellow regions



(a) Browser app (b) Chat app (c) Music app (d) Board game

Figure 7: Illustration of touch fingerprint patterns identified for different applications.

in the CNN filter output represent the touch events shown in Figure 6b. By concatenating the features from both filters, the network is able to suppress noise and accurately extract the touch data.

Sequence Mapping. The extracted features are used for the sequence mapping to extract the binary touch data. We use two layers of bi-directional LSTM to learn the binary sequence. Each sample at a time instance, t, in the sequence depends on the previous and future samples. We design the network to predict the binary label, $z_{t/2}$, of the middle sample, $s_{t/2}$, at time instance, $t/2$, given a window of samples from s_0 to s_t in the time interval $[0, t]$. We define the classifier as follows:

$$z_{t/2} = \underset{c_i \in \{0,1\}}{\operatorname{argmax}} P(c_i | s_0, \ldots, s_t) \tag{1}$$

This design enables the network to use the preceding and future samples for each output. To compromise between the network complexity, processing time, and sequence length, we use a sliding window with a size of three seconds as network input, which captures the dependence between the sequence samples. The final layer is a dense layer with a softmax activation function that classifies the middle sample for each window as 1 if a touch is detected or 0 otherwise. We detail our network architecture in Appendix A.

4.3 Application Classifier

The leakage of touch events can be exploited in several ways. In iStelan, we show the feasibility of detecting the app type a user is using from the pattern of touch events. We found that the usage patterns of different app types are unique and often repeat based on the app functionality. Specifically, every time an app is used, the user repeats a sequence of actions depending on the app type, which results in a discriminating touch pattern.

Figure 7 shows an example of touch patterns for different apps. In a browser app (Figure 7a), a user starts typing text in the search bar, which appears as dense and narrow touch events. The user then scrolls over the search results, giving more sparse and wider touch events. Lastly, the user selects a page and remains idle for some time to read. The chat app example (Figure 7b) starts with the user selecting a chat room. The user then alternates between typing, sending, and receiving chat messages. In a music app (Figure 7c), the user browses the available songs, selects a song, and remains idle while listening. Occasionally, the user would change the volume, press stop, replay, etc. A board game (Figure 7d) is usually characterized by many small similar touch events.

Overall, we observe that each app results in a different touch pattern, which enables iStelan to distinguish the app type. Several patterns may exist for the same app; however, these patterns are still limited by the possible user activities on the app. To account for this, we collect samples from multiple users, which allows diversity in the training dataset (Detailed in Section 5).

Application Classifier Design. We predict the app type from a given sequence of binary touch data. First, we encode the binary touch data into a feature vector, then feed it into a classifier to detect the app type. The number of app types to be detected could vary based on the attacker's target and interests. Therefore, we leverage the one-vs-all multi-class classification technique, where we build a two-class classifier model for each app type. We train each model using the data from all apps. For each model of an app type c, the samples are labeled as 1 for c and 0 for all other apps. To accommodate for class imbalance, we use weighted samples for each class label. Specifically, each model has a binary output, which gives 1 if the app's pattern is detected and 0 otherwise.

The output of these models is aggregated using a max function, where an app pattern with maximum probability is selected as the detected app type. In some cases, the patterns could be very noisy. In these cases, all app-type models will output 0 where none of the models detect a valid pattern. The attacker could discard such samples and wait for the target apps to be detected.

We use an LSTM network for each app-type model, which takes a sequence of encoded touch data as input to capture the touch pattern specific to an app. LSTM captures the temporal aspects of the touch sequences using feedback connections. We compared with other techniques in Appendix D. We define the length of the input sequence as the app usage window size, w_s. We found that a w_s up to 60 seconds better generalizes to new subjects and yields more accurate classification (evaluated in Section 5).

Since the magnetometer sampling rate is 100Hz, a window size of 60 secs results in a sequence of 6000 samples. This input is relatively large to be learned using the LSTM network, which may incur high computation time in the background at inference time. To address this problem, we apply integer encoding to the binary sequence of touch events by counting the number of consecutive 0's and 1's. A continuous patch of n 1's is replaced by an integer k^+, where $k^+ = n$. A continuous patch of m 0's is replaced by an integer k^-, where $k^- = -m$. Integer sequences are padded with zeros to achieve equally sized sequences fed into the LSTM network.

4.4 Touch Event Type Detector

iStelan labels touch events with touch types given the extracted binary touch data and the predicted app type. This adds another stage to privacy-threatening inference, where it enables detecting and analyzing the user's activity on the app. We infer four different touch event types: `tap`, `swipe`, `type`, and `long press`.

Since the different touch event types could be similar in the binary sequence, e.g., a long press and a short swipe, we cannot trivially predict the touch type from its length. To address this problem, we model the touch event labeling problem as a statistical Markov model with HMM [45] where the observations are extracted from the binary touch data, and the touch types are the hidden states. The sequence of touch events depends on the activities performed on an app, i.e., differs with app functionality. Thus, we build a separate HMM model for each app type to capture the dependencies between touch event types.

We estimate the HMM parameters from a training dataset for each app. When a sequence of binary data from a specific app arrives, iStelan extracts the observations and uses the pre-computed HMM model parameters to estimate a sequence of touch event labels. HMM is selected for this task due to its effectiveness on sequence data, where it models the data as a state machine and uses statistical models to predict hidden states. We compared using a baseline HMM with fixed parameters that depend only on the frequency of labels for each app type. We also evaluated using one model for all app types instead of a separate model per app. The proposed HMM model with dynamic probabilities and separate models shows the best performance since the touch sequences vary for each app type (Detailed in Appendix D).

We extend the Viterbi algorithm [8] to compute the maximum likelihood sequence (MLS) of hidden states for the current window using dynamic programming.

Touch-Type Extractor. The input to the HMM model is a sequence of window size (w_s) of binary touch data. A set of observations $Y = (y_1, \ldots, y_M)$ is extracted from the binary data, where $y_m = (a, b)$ for $1 < m < M$ is a pair of integers, a is the length of a touch event, and b is the length of idle time preceding the touch event.

Let $S_m = s_{1,m}, \ldots, s_{N,m}$ be the set of possible states for an observation y_m, where N is the number of possible touch event types. To infer touch-type labels, we use three probability distributions:

- The observation probability distribution of y_m at state $s_{i,m}$ is defined as $B = P[y_m|s_{i,m}]$. This represents the probability of observing input y_m given the actual touch event label $s_{i,m}$.
- The state transition probability distribution from a state s_i to a state s_j. We define it as $A = \{a_{ij}\}$, where $a_{i,j} = P[s_j|s_i]$.
- The initial state probability, $\pi = \{\pi_i\}$, where $\pi_i = P[s_{i,1}]$. We assume it is uniform among all states.

iStelan finds the most probable sequence of touch event types, S, from a given sequence of touch event observations, Y. We use the Viterbi algorithm, which uses dynamic programming to infer the hidden states sequence, S^*, modeled as $S^* = \text{argmax}_S P[Y|S]$.

Touch-Type Parameter Estimator. We use a training dataset, Ω, to estimate the parameters to compute the observation and transition probabilities. To compute the observation probability of $y_m = (a, b)$, we first compute the frequency f_i of observing a state s_i in Ω. Given the total number of states in Ω is f, the observation probability depends on f_i/f. Second, the observation probability depends on the event duration a. For example, a short event is more likely to be a tap rather than a swipe. For each state, we assume the event duration has a Gaussian distribution around a mean μ_a. Therefore by combining these factors and assuming independence, we model the observation probability as:

$$P[y_m|s_{i,m}] = \frac{f_i}{f} \cdot \frac{1}{\sqrt{2\pi}\sigma_a} e^{-0.5(\frac{a-\mu_a}{\sigma_a})^2} \quad (2)$$

where μ_a and σ_a are the distribution mean and standard deviation of event duration for state $s_{i,m}$ estimated from Ω.

Similarly, the transition probability from state s_i to state s_j depends on two factors: the frequency of transition from s_i to s_j in the training set and the idle duration b between these transitions. Therefore, to obtain the transition probability $a_{i,j}$, we compute the frequency $f_{i,j}$ of transitions from s_i to s_j and the frequency f_i of all transitions from state s_i using Ω. The idle duration b between touch events is also modeled as a Gaussian distribution. Assuming independence, we define the transition probability as:

$$P[s_j|s_i] = \frac{f_{i,j}}{f_i} \cdot \frac{1}{\sqrt{2\pi}\sigma_b} e^{-0.5(\frac{b-\mu_b}{\sigma_b})^2} \quad (3)$$

where μ_b and σ_b are mean and standard deviation of idle duration between states s_i and s_j estimated from the dataset Ω.

5 EVALUATION

We report our experience of applying iStelan attack on data collected from 22 users in total. We analyze its effectiveness at each attack stage in inferring users' privacy-sensitive data. Our study on 17 users shows that iStelan yields an average accuracy of 90% in uncovering users' touch events, 74% in inferring app types, and 73% in identifying the touch events of users. We also collected data from 5 users to evaluate iStelan with different samples of application types or when the application type is unseen.

In addition, we conduct experiments to show iStelan's performance under different phone positions (e.g., when the phone is placed on a table and when the user holds the phone in hand while sitting/walking) and environmental factors (e.g., when the user wears a metal object and uses the phone when people are around).

To the best of our knowledge, no prior work have presented an end-to-end combination of touch event identification from magnetometer signals, app type classification from binary touch sequence, and touch event label extraction through HMM. Thus, in Appendix D, we perform a per-attack stage comparison against techniques of prior work. We also compare iStelan with recent side-channel attacks that detect touch events based on accelerometer and gyroscope motion sensors. iStelan yields higher accuracy than previous approaches and is more effective at detecting touch events than using the accelerometer and gyroscope. We present our results by focusing on several research questions:

RQ1 How effective is iStelan in detecting the binary touch events, app types, and touch event labels?

RQ2 What is the accuracy of iStelan per app?

RQ3 How does iStelan perform across different users?

RQ4 How does iStelan perform at different sensor sampling rates?

RQ5 How robust is iStelan's app classifier?
RQ6 What is the impact of different environment factors on iStelan's performance?
RQ7 How does the phone position affect iStelan's performance?

Evaluation Setup. We evaluated iStelan on iPhone 10 XS device running iOS version 14, which is provided to the participants in the user study. We implemented an app that runs in the background, acting as a malicious app, which mimics a legitimate fitness tracker that has access to the magnetometer sensor and stays alive in the background by accessing location services.

We selected seven open-source apps from the most popular iPhone app categories on the app store[2]. These apps include a board game, browser, music, chat, maps, shooting game and shopping. These apps are similar in their functionality and design to the popular apps found on the Apple store, i.e., 2048 Puzzle game, Chrome browser, Spotify, Whatsapp, Google Maps, Classic Arcade game, and Amazon shopping. Additionally, we conducted experiments on 14 apps from the same 7 app types, where we tested on two new samples per app type.

We also collected data from 5 new app types, including ToDo, Finance tracking, Instagram, Weather, and Text Editor, to evaluate iStelan performance on unseen app types. The app details are provided in Appendix B. The malicious app collects the magnetometer data at 100Hz in the background. For ground-truth, we record the touch API data from each app when used in the foreground. The malicious app consumes, on average, less than 3% of a fully charged battery while logging the magnetometer data over 2 hours.

During our experiments, we did not restrict background applications or services. While performing each experiment, a participant used 3 to 4 apps, which could be open in the background. The phone used in the experiments has approximately 50 apps allowed to operate in the background. These apps are likely to have background services running during the experiments.

We train the models on a desktop with a 2.4GHz 6-Core Intel Xeon E5-264 processor with 32GB RAM and Nvidia Quadro M4000 GPU with 8GB RAM, using Keras with Tensorflow backend.

User Study and Data Collection. We performed a user study to evaluate the effectiveness and performance of iStelan. We collected data from 17 subjects recruited at the university campus, including 5 females and 12 males. We asked 8 subjects to use board game, browser, chat, and music apps (**Apps_A1**); the other 9 subjects to use map, shooting game, and shopping apps (**Apps_B1**). We collected more data from 5 additional male subjects using different app samples (**Apps_A2, A3, B2, B3**) and unseen app types (**Apps_C, C'**). All subjects are graduate or undergraduate students. The app sets used by each subject are given in Appendix B.

Although subjects were already familiar with the popular app types used in our experiments, we show them how to start the apps due to the slight variations in our app samples. We then asked each subject to use different apps as they typically use them in real life, while magnetometer data is collected in the background. A subject uses each app for 10 minutes. Subjects use the phone when placed on the table, in hand while sitting and walking, depending on the conducted experiment type (Section 5.1). Each app sample duration is one minute. Subjects were allowed to switch between apps after each sample. To evaluate different use cases and environment scenarios, we asked a set of users to intentionally wear metal objects or use the phone in a lab, at home, or in a hall when people are around. To demonstrate iStelan's effectiveness when a user switches apps, we performed experiments where a subject uses an app for an arbitrary time and then switches to another app. In total, we collected over 13 hours of data. A pilot experiment was initially done on one user for parameter tuning. Classifiers were trained for up to 50 epochs to select the best models.

[2]https://www.statista.com/statistics/270291/popular-categories-in-the-app-store/

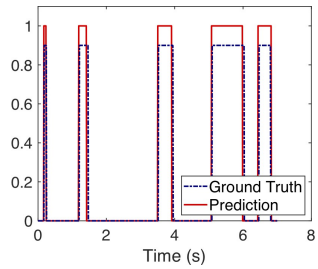


Figure 8: A trace example of `Stage-1` output. The predicted trace (red, solid) follows the ground truth (blue, dashed).

We employ the leave-one-out method to evaluate the performance of each attack stage of iStelan. Specifically, we pick each user once for testing as the unseen user, while the rest of the users' data is used for training the models. This means an attacker can collect data from a set of users to train iStelan models and test them with unseen users. We repeat this process for all users and then report the average performance metrics for each attack stage.

Ethics. We obtained approval from the Institutional Review Board (IRB) for our user study. We advertised our study through the internal campus email listings and Slack channels. Participants are asked to sign a consent form where we explain the study details and data collection. Participants are informed that their participation is voluntary and they have the right to quit the experiment at any time. During the experiment, we provide the smartphone to the participant and verbally explain the user study procedures. We ask the participants not to enter any personal data. We do not collect sensitive data except touch event duration and magnetometer data. The data is stored securely and confidentially on our servers.

5.1 Effectiveness

The experimental results reported here were conducted in a CS lab environment where electrical devices such as monitors, computers, and laptops were present in the surroundings. Each user used a phone placed on a table in the lab and was wearing their everyday clothes, including jackets with metal zips, phones, and keys in pockets or backpacks around them. We present other experimental scenarios in Section 5.3.

iStelan Performance (RQ1). We evaluate the accuracy of each stage of iStelan on the data collected from 17 users while using app sets **Apps_A1** and **Apps_B1**. We refer to the three stages of our attack as `Stage-1`, `Stage-2`, and `Stage-3` for binary touch extraction, app classification, and touch event type detection.

Figure 8 shows an example of an extracted binary touch trace from a user compared to the ground truth touch data collected through iOS touch API. We multiply the ground-truth data by a

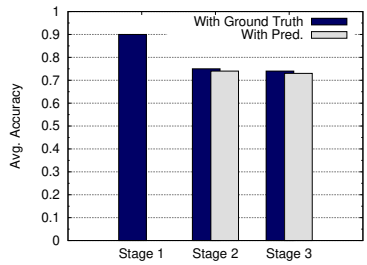


Figure 9: Average accuracy of iStelan attack stages. For `Stages` 2 and 3, the blue and grey bars show the accuracy using ground truth and predicted binary touch data.

factor of 0.9 for visualization. The predicted touch event sequence (output of `Stage-1`) accurately follows the ground truth touch data. We compute the accuracy by comparing every bit of the ground truth and the prediction traces using the leave-one-out method and averaging among all users. Figure 9 shows that iStelan extracts users' touch (`Stage-1`) with an average 90% accuracy. This corresponds to 88% recall, 90% precision, and 89% F-measure.

To show the impact of our binary touch extraction module on the following stages, we compute the accuracy of app classification (`Stage-2`) and touch event type detection (`Stage-3`) (a) using the ground truth binary touch data, and (b) using the predicted binary touch data from `Stage-1`. For app classification (`Stage-2`), we compute the percentage of correctly classified apps averaged among all users. Figure 9 shows that the average accuracy for app classification is 75% using ground truth data and slightly decreases to 74% using the predicted data from `Stage-1`. These results show that iStelan is effective for an end-to-end real-time privacy attack.

The output of the touch-type detection is a sequence of labeled events. Binary extraction (`Stage-1`) errors introduce false-positives (FPs) where touch events not in the ground truth are falsely detected and false-negatives (FNs) where true touch events in ground truth are not detected. Due to these FPs and FNs in `Stage-1`, the labeled ground truth touch sequence and the predicted sequence using the binary extracted data may not perfectly align. Therefore, to evaluate `Stage-3`, we first perform a sequence matching between ground truth and predicted sequence labels. We use the percentage of true positive (TP) labels detected from the ground truth sequence for accuracy. We ignore the FPs of `Stage-1`; since they do not map to corresponding labeled events in the ground truth.

We define the number of correctly labeled events as E_c and the length of the labeled ground truth sequence as L_{GT}; thus, the accuracy is defined as E_c/L_{GT}. Figure 9 shows that the touch-event type detector (`Stage-3`) achieves, on average, 73% accuracy using both ground truth and predicted binary touch events, making it feasible to learn user's touch behavior within apps.

Per-Application Accuracy (RQ2). In this set of experiments, we evaluate the accuracy of each attack stage of iStelan for different apps. As shown in Figure 10, the binary touch extraction accuracy (`Stage-1`) is consistent across different apps.

We next present the accuracy of the app classifier (`Stage-2`) per app. We note that our one-vs-all classifier might fail to detect any app pattern, i.e., the output of all app models is 0. In our dataset, only 7% of the samples are not classified to any app. We consider these as noisy samples and discard them from further evaluation.

The confusion matrix of the app classifier (`Stage-2`) is shown in Figure 11. We found that the board game app is classified with the highest accuracy, given its distinct touch fingerprint patterns. Since browser and map apps often have similar touch activity (e.g., if a user searches for a location on a browser or a maps app), this leads to some confusion between the two apps. Despite this confusion, the touch pattern revealed by iStelan can be used by an attacker, e.g., to recommend a maps app to users using maps on a browser.

Lastly, the accuracy of the touch event type detector (`Stage-3`) for different apps is shown in Figure 12. The board game app has the highest accuracy due to the high frequency of swipes and taps. The average accuracy is between 61%-75% for other apps.

Per-User Accuracy (RQ3). We study whether iStelan's performance is affected by different users. Figure 13 shows the performance of three stages of iStelan for 17 users. We note that in all our experiments, the users were wearing their everyday clothes, including jackets with metal zips, and having their phones and keys in their pockets or backpacks. Each user was asked to use apps as they would typically use them in real life. Despite this variability, the system shows consistent results for the three stages across different users. We observe standard deviation of 0.03% for `Stage-1`, 0.1% for `Stage-2` and 0.06% for `Stage-3` across users.

Impact of Sampling Rate (RQ4). One defense approach to mitigate sensor-based side-channel attacks is to reduce the sensors' sampling rate for background apps. Therefore, we evaluate how iStelan performs under varying sampling rates. Figure 14a shows iStelan's (`Stage-1`) accuracy at different sampling rates ranging from 10Hz to 100Hz. We observe slight accuracy differences at different sampling rates due to the changes in the data sequence length at different rates, causing changes in the features and model weights. However, with all sampling rates above 10Hz, iStelan's accuracy is consistently greater than 88%, which shows that the changes in sampling rate are not effective against iStelan attack.

5.2 Robustness of Application Classifier

We evaluate the robustness of the app classifier **(RQ5)** by varying its window size parameter, testing when the user switches between apps, and when tested on an app type with a different UI.

Window Size Parameter. The app classification takes the window size (w_s) of the touch data as an input. We evaluated the app classifier's performance using different window sizes within the range of 10 to 60 secs. As shown in Figure 14b, iStelan correctly infers the app type when the window size increases. The best accuracy is observed when a victim spends at least 40 secs on an app. If the victim spends 20 secs using an app, iStelan is able to identify the app type with 70% accuracy. Considering an average time of 20-40 secs, this leads to on average 74% accuracy.

Application Transitions. We evaluate the accuracy of iStelan when the user uses an app for an arbitrary time and then switches to another app. We asked a user to use one app for an arbitrary interval between 30-60 secs, then switch to another app for another interval between 30-60 secs. We repeat for all the combinations of transitions between 4 apps. Each experiment is repeated 3 times for each transition. We use the default classifier model trained when

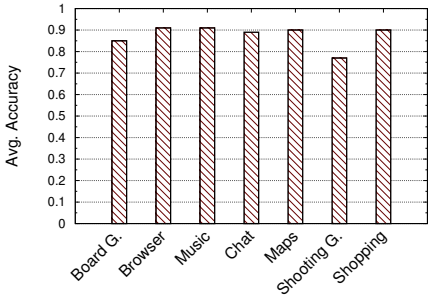


Figure 10: Accuracy of binary touch extraction for different apps (`Stage-1`).

Figure 11: Confusion matrix of the app classifier (`Stage-2`).

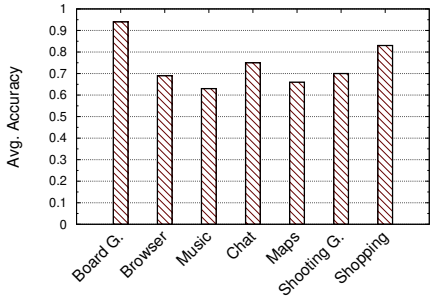


Figure 12: Touch event type detection accuracy for different apps (`Stage-3`).

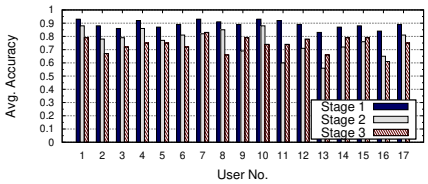
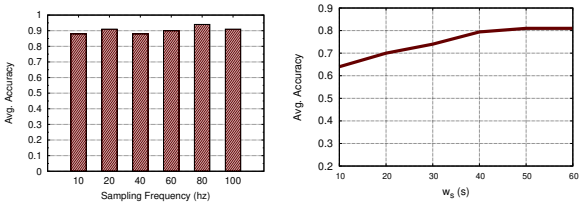


Figure 13: Accuracy for each user in different attack stages.



(a) (b)

Figure 14: Impact of (a) magnetometer sampling rate on binary touch extraction accuracy (`Stage-1`), and (b) window size (w_s) on app classification accuracy (`Stage-2`).

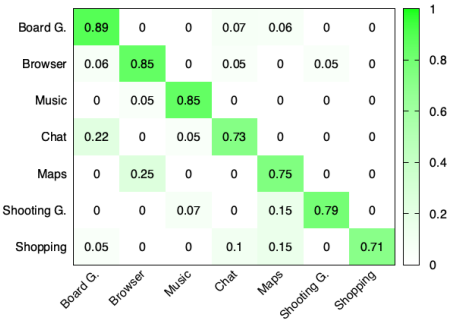


Figure 15: App classifier's confusion matrix on apps with different UI (`Stage-2`).

using one app at a time and test using the transition data. Since the start of the app is unknown, we use a sliding window of 40 secs (see Figure 14b) with a step of 5 secs to allow for pattern changes in each window. Table 3 demonstrates that iStelan correctly classifies the apps before and after the transition phase. The accuracy slightly decreases to 77% during the app transition as the user performs quick swipes, which iStelan does not separate from the app usage pattern. However, since the app transition phase is relatively small, on average 6.36 secs, iStelan quickly detects the app that the user switches to after the transition phase ends.

Applications with Different UIs. We evaluate how iStelan's app classifier (`Stage-2`) performs when trained and tested on a different set of app samples from the same categories. Since iStelan, for example, can correctly classify the chat app category, it should be able to classify a different chat app. For this purpose, we conducted an experiment where two subjects used two new samples from each app type. For the board game, browser, chat, and music app types, the two new samples are given in sets **Apps_A2** and **Apps_A3**. While for the maps, shooting, and shopping app types, the two new samples are given in sets **Apps_B2** and **Apps_B3**. These are open-source apps similar in their functionality to the training apps but with a different UI (e.g., Chrome browser and Mozilla browser).

We used the data from previous experiments, collected from 17 subjects using **Apps_A1** and **Apps_B1** for training. We tested on data collected from two subjects while using the new set of apps. Figure 15 shows the confusion matrix for the app classifier when tested on these unseen apps. iStelan accurately classifies an app type even if the UI is different due to the similarity in the extracted touch patterns for apps in the same category.

Unseen Application Categories. While an attacker trains a classifier using app types of interest, touch patterns from unseen apps, i.e., not in the training set, could also be collected. The attacker needs to detect the unseen app types and discard them.

Intuitively, unseen app types should be classified as negative by all of the binary app classifiers. However, these supervised classifiers build a decision boundary between positive samples (from one app type) and negative samples (from the remaining 6 app types) during training. During testing, a new sample from an unseen app type is not guaranteed to be correctly classified. Therefore, an additional layer is needed to detect and filter the unseen types.

We use the Local Outlier Factor (LOF) algorithm for novelty detection [6]. LOF is a semi-supervised technique trained using our initial training data as positive samples. The goal is to detect whether a new sample is an outlier (negative sample). The LOF algorithm finds anomalous data points by measuring the local deviation of a given data point with respect to its neighbors. A point with a lower density than its neighbors is an outlier.

Table 3: iStelan accuracy with application transition.

Detection interval	Average App Classification Accuracy
Before & after transition	0.81
During app transition	0.77
Overall accuracy	0.80

Table 4: Effect of environment changes on attack stages.

Environment Scenario	Stage 1	Stage 2	Stage 3
Wearing a metal object	0.86	0.79	0.72
At home environment	0.93	0.82	0.83
In hall with people around	0.92	0.86	0.86

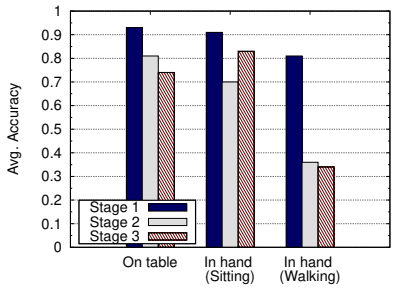


Figure 16: Accuracy with different phone positioning.

We collect data from two participants using five new app types (ToDo, Finance tracking, Instagram, Weather, and Text-Editor) from sets **Apps_C** and **Apps_C'**. We train the model on the initial data of the 7 app types from **Apps_A1** and **Apps_B1**. For testing, we collect data from the same 7 app types from two participants using sets **Apps_A2** and **Apps_B2**, to be used as the positive test samples. We use the data from the 5 unseen apps as negative samples. iStelan detects the unseen apps (negative samples) as outliers with an accuracy of 78% while the known apps (positive samples) are detected as inliers with an accuracy of 72%. These errors are due to the similarity in touch patterns from some of the unseen apps with the apps used in training, e.g., finance and shopping apps.

Without the novelty detection layer, the binary classifiers only detect 29% of the unseen apps. We also tested another approach where we train a separate novelty detection model for each of the 7 app types. A new sample from an unseen app-type is fed into the 7 models and should be detected as a negative sample from all models. This technique achieves similar accuracy for negative samples (78%) while a lower accuracy for positive samples (64%). The reason is that the number of samples per app is relatively low, making novelty detection more challenging. We also applied a set of other popular novelty detection algorithms, frequently used for sequential data: One-class Support Vector Machine [57], k-Nearest Neighbors [18], K-means [10], Isolation Forest [27] and Least-squares Probabilistic Classifier [44]. We find that the accuracy of these algorithms is lower compared to the LOF algorithm since it provides a better representation of the high-dimensional touch features and considers sample densities in measuring the distance between samples from known and unseen apps.

5.3 Environmental Factors

Effect of Different Environmental Factors (RQ6). The previous experiments were conducted in a CS lab setting where electrical devices (e.g., monitors, computers, and laptops) were present. The presence of these devices does not impact the iStelan performance. However, other environmental factors (e.g., non-static metallic objects) could affect the magnetometer data.

To evaluate the impact of the ambient magnetic field on iStelan's performance, we conducted two additional experiments with one subject in a home environment (`setting-1`) and with one subject in a hall with people present in the surroundings (`setting-2`). Additionally, we evaluate how iStelan's performance is affected by the presence of metallic objects in the vicinity of the phone. For this purpose, we asked a subject to perform experiments in the lab setting while wearing a metallic bracelet on their wrist (`setting-3`).

Table 4 summarizes the accuracy of iStelan for each of the described scenarios. We use the model trained in the default settings to evaluate these scenarios. iStelan performs consistently across environments with different ambient magnetic fields. When the user wears a metallic bracelet, we observe little magnetic interference from the bracelet, resulting in an average accuracy of 86% for binary touch extraction compared to 90% accuracy in the default setting without the bracelet (RQ1). Yet, the metallic bracelet's magnetic impact is much smaller than the magnetic impact of the user's touch despite its closeness to the smartphone. Therefore, iStelan can accurately extract the touch data and use it for app classification and touch event type detection, regardless of this interference.

Effect of Phone Position (RQ7). The magnetic field measured by the magnetometer can be affected by how the user uses the phone. The experiments in Section 5.1 were conducted with the phone placed on a table to show that iStelan does not require movement of the phone to detect touches. To evaluate the performance of iStelan with different phone positioning, we conduct two additional experiments with one subject in each: (a) phone in hand while the user is sitting, (b) phone in hand while the user is walking.

Figure 16 shows the accuracy of iStelan with different positioning. We found that accuracy is consistent when the phone is on a table or in hand with the user sitting, which shows the applicability of iStelan attack with widely used phone usage scenarios. When the user is walking, iStelan still achieves an accuracy of 81% for binary touch extraction (`Stage-1`). However, the accuracy becomes low for `Stage-2` and `Stage-3`. The main reason is that the ambient magnetic field changes rapidly when the user is walking, which mitigates the magnetic effect of the touch. This was also proven by the previous works [2, 52] that use sensors to infer touches, where none of these works achieve good results with the user in motion scenarios, as detailed in Section 2.

6 LIMITATIONS AND DISCUSSION

6.1 Countermeasures

In order to mitigate iStelan's attack, the design of touch screens in iOS devices can be changed or improved to isolate their magnetic effect, which could prevent information leakage. However, design changes are costly and time-consuming, and modifying existing mobile devices would be more expensive. Thus, we describe several countermeasures to mitigate or prevent iStelan attack.

The first defense, *noise injection*, hides the capacitive screen effect on the magnetometer by injecting noise into its signal. The noise must interfere with the small effect on the magnetometer due to

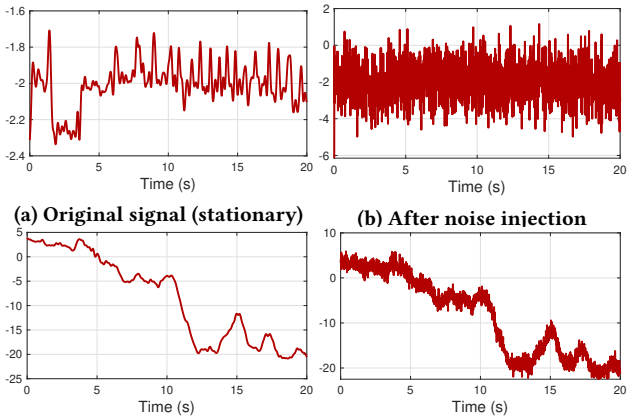


Figure 17: Effect of noise injection on the magnetometer signal: (a)-(b) when the phone is stationary and (c)-(d) when the user is holding the phone and walking.

touches while still guaranteeing correct functionality for legitimate apps that use the magnetometer. We implemented this defense technique by adding Gaussian noise to the magnetometer signal with a small signal-to-noise ratio, SNR = 0.3.

Figure 17 shows the effect of signal injections. The added noise clearly hides the touch effects when the phone is stationary. However, when the user is walking, the small fluctuations due to the touch are hidden, while the overall trend of the signal is still the same. Our finding highlights that this technique would not harm apps that require coarse-grained magnetometer data for operation, such as navigation apps. However, in apps where more fine-grained information is required, e.g., as in gaming apps, the correct functionality of the app is affected by this noise injection. Moreover, it is difficult for the vendors to implement this defense to find complete test scenarios to ensure app functionality is not affected.

The second defense is the *malicious app detection*, where the OS can monitor apps' behavior and flag apps that potentially deviate from their normal behavior [26, 46, 51], e.g., apps that periodically send collected data to a remote server at a high rate. However, profiling the expected behavior of apps often leads to high false alarms [1]. Other defenses, such as App-Guardian [64], propose to detect malicious apps on the application level without modifying the OS. It works by pausing suspicious background apps when a protected app comes to the foreground. It then resumes the background apps after the protected app finishes its tasks. App-Guardian learns side-channel information of apps to infer their suspicious activities, including the name of a service thread, a thread's scheduling status, and the amount of kernel time it consumes. iStelan only collects magnetometer sensor data in the background, which is considered insensitive data by iOS; thus, it could act totally as a legitimate app (e.g., a fitness tracking app). However, due to the potential side channels of motion sensors, App-Guardian might consider sensor data as sensitive and, therefore, flag iStelan's app as malicious. Yet this approach, as acknowledged in their limitations [64], can terminate legitimate apps affecting their utility. This brings a level of inconvenience to the user who could experience delays or stop background services. Therefore, [64] suggests that more accurate identification of malicious activities is required to balance the tradeoff between privacy and apps' utility.

The last defense technique, *restrictive methods*, restricts background access to the sensors by the OS [20, 43]. However, this would negatively affect the functionality of several apps collecting magnetometer data for their required operations. The iOS could also add explicit permissions for sensors to increase restrictions over apps accessing sensors; however, most users grant permissions since they are unaware of the threats behind motion sensors [42].

6.2 Improving iStelan Accuracy

Large-Scale Data Collection to Improve Generalization. We evaluated iStelan on a total of 22 users on 7 app types. Each user performs 10 experiments per app. In order to improve the attack accuracy, an attacker could perform a large-scale data collection from more users on a larger number of apps, with multiple app samples per app type. This may help increase the generalization of the model by covering a variety of usage patterns for each app type and properly improve its ability to adapt new test data.

Using Different Data Modalities. An attacker may also combine different side channels that exploit magnetometer data to improve iStelan's accuracy. For example, other techniques for app fingerprinting leverage patterns in magnetometer data resulting from varying CPU workload, battery, or app usage [31, 40]. While these techniques could be prone to high errors due to app variations and background noise (shown in Appendix D), combining the fingerprint data from different side channels as inputs of a model or using multiple learning algorithms for each side channel and ensembling their results may improve the detection results of iStelan.

More Sophisticated ML Algorithms. iStelan's accuracy could also be improved using more sophisticated ML algorithms. For instance, contrastive learning could be extended for outliers detection [19] to reduce the impact of the noisy samples on the iStelan's overall accuracy, and for supervised learning that makes the same class samples (app types) to be pulled together while different classes are pushed apart [24]. Other types of ML algorithms, such as auto-encoders and attention mechanisms [58], could also be integrated into iStelan for touch extraction and application classification for improving accuracy. However, direct application of these approaches to iStelan requires large-scale data collection and prior knowledge of task-dependent and specific invariances [60].

Domain-Specific Attacks. iStelan could also be exploited by an adversary to launch different attacks with relatively low confidence or increase the confidence in existing attacks. For example, an attacker could train a classifier to detect specific application types that are popular in certain locations and further increase confidence by increasing the samples of this particular type. This app type can be used to launch GUI-confusion attacks, allowing the attacker to steal data from the victim phones. This will enable inferring sensitive information of several victims even if the attack confidence is low. The attacker can also target specific victims to learn their behavior and infer their sensitive data. Another potential attack is to track users' behavior across different apps, which became more crucial after Apple added the app tracking transparency (ATT) permission. Due to this permission, app developers are trying to find potential ways to fingerprint users' phones and infer their

behaviors on different apps [25]. iStelan could be integrated with other data to increase the probability of detecting users' behavior, allowing more personalized ads and increasing revenue.

6.3 Limitations

We demonstrated that changes in the environment, including different locations or the presence of nearby metal objects, do not affect the touch detection model. Yet, detecting the small fluctuations in touch events would be challenging in some scenarios, such as while a user is walking, in a vehicle, or in an elevator, since the ambient magnetic field changes rapidly.

iStelan could be extended to use a more sophisticated filtering step to address these scenarios. For example, the system could first detect when the user is moving using a combination of motion sensors, then launch the attack when the device is stationary. We note that side-channel attacks exploiting motion sensors for inferring touch data also share this limitation, detailed in Section 7.

iStelan is able to accurately detect whether a touch event occurs; however, it does not detect the exact touch location, which can be used to infer a user's keystrokes. Our experiments showed that detecting the touch location in practical scenarios is challenging since the impact of touch on the magnetometer slightly varies across different screen locations. Future work will explore the intensity of the magnetometer fluctuations with additional features from other built-in sensors to infer touch locations.

Lastly, we show that iStelan can detect various app types with different features; however, if the app is very complex with several complicated features, the touch patterns would be random and not be easily detected by the app classifier. However, this behavior is unlikely to appear in popular apps, where app designers target simplicity and repetitiveness to increase the usability of apps.

7 RELATED WORK

Touch Behavior Inference via Mobile Sensors. Recent efforts have explored potential data leakage by inferring users' touch behavior from smartphone touchscreens. ACCessory [38] demonstrated that typing on smartphone touchscreen results in accelerometer fluctuations, which leaks sensitive user information. However, this attack works when the user holds the device in landscape using both hands and enters text using her thumbs. TouchSignatures [34] exploited motion sensor data collected through malicious JavaScript code on web browsers to identify touch actions and infer user PIN codes. Similarly, TapLogger learned the patterns in motion sensor data to infer different tap events [62].

These works rely on a noticeable movement in the phone body to detect touch actions via the accelerometer/gyroscope data and add restrictions that hardly apply to many smartphone use scenarios in practice. In contrast, iStelan does not rely on any smartphone movement. It exploits the electromagnetic touch impact on capacitive smartphone screens, even when the phone is completely stationary or freely used, covering more practical scenarios.

Side-channel Attacks via Electromagnetic Sensing. Recent efforts have unveiled several side-channel attacks based on electromagnetic data collected by smartphone magnetometers. Recent works [4, 12] showed that the electromagnetic effect of a computer drive could be exploited by the magnetometer of a nearby smartphone to infer system details or user activities. These works rely on close proximity between a user's smartphone and computer and leak information about apps used on a computer.

Electromagnetic sensing has also been exploited for eavesdropping on user handwriting by analyzing the magnetic field changes of stylus pens [16, 28]. However, they require the attacker device to be placed within close proximity of the target device and are limited to stylus pens with embedded magnets. In contrast, iStelan leverages the small impact of human touch on the onboard magnetometer to infer privacy-sensitive information.

Another line of work demonstrated that electromagnetic emissions from smartphone processors could reveal information about ongoing activities, allowing device identification [41] and website/app fingerprinting [31]. However, they make impractical assumptions about device usage (e.g., no background activity and only one tab open in a browser). Moreover, the fingerprints of apps rely on specific configurations, which are unknown in practical scenarios. MagThief [40] also proposed to detect app usage information from the EM signals emitted during the execution of app-related tasks. However, this work is also highly prone to variations due to background activity noises and ambient magnetic noises. A recent work [37] observed a correlation between the color emitted by the phone LED display and magnetometer data and used a DNN to infer users' app usage. Yet, it assumes apps have different splash screen colors and is affected by app performance or changes in UI. In contrast, iStelan is robust to UI changes and does not rely on app performance or phone configurations. Other works exploited the built-in magnetometer for eavesdropping on Magnetic Secure Transmission in mobile payments, e.g., Samsung Pay [13].

Recent works [30, 50, 59] have also exploited capacitive screens to induce touch events through EMI interference on a victim's smartphone. However, they require external hardware to be placed in the user's vicinity. iStelan exploits leaked touch data remotely to infer the user's activity without using any extra hardware.

8 CONCLUSIONS

In this paper, we present a new type of side-channel leakage in iOS devices; the electromagnetic effect of capacitive touch screens is observed through the onboard magnetometer sensor. This side-channel eliminates the restrictions of prior works on how users hold their phones and detects users' touch activity both when the device is stationary and naturally held in hand. We introduce iStelan, which exploits this effect to expose users' sensitive data, including their touch events, the app type they are using, and touch event types within the app. We conduct extensive experiments to show the practicability of iStelan attack with data collected from 22 users and evaluate its performance in different scenarios.

ACKNOWLEDGMENTS

We thank our shepherd and the anonymous reviewers for their comments and suggestions. This work has been partially supported by the National Science Foundation (NSF) under grant CNS-2144645 and startup funding from Purdue University. The views expressed are those of the authors only.

REFERENCES

[1] Yasemin Acar, Michael Backes, Sven Bugiel, Sascha Fahl, Patrick McDaniel, and Matthew Smith. 2016. Sok: Lessons learned from android security research for appified software platforms. In *IEEE Symposium on Security and Privacy (IEEE S&P)*.

[2] Adam J Aviv, Benjamin Sapp, Matt Blaze, and Jonathan M Smith. 2012. Practicality of accelerometer side channels on smartphones. In *Annual Computer Security Applications Conference (ACSAC)*.

[3] Antonio Bianchi, Jacopo Corbetta, Luca Invernizzi, Yanick Fratantonio, Christopher Kruegel, and Giovanni Vigna. 2015. What the app is that? deception and countermeasures in the android user interface. In *IEEE Symposium on Security and Privacy (IEEE S&P)*.

[4] Sebastian Biedermann, Stefan Katzenbeisser, and Jakub Szefer. 2015. Hard drive side-channel attacks using smartphone magnetic field sensors. In *International Conference on Financial Cryptography and Data Security*.

[5] J.P.G. van Brakel. 2014. Robust peak detection algorithm using z-scores. https://tinyurl.com/5bv9p4sy. (version: 2020-11-08).

[6] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *ACM SIGMOD international conference on Management of data*.

[7] Liang Cai and Hao Chen. 2011. TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone Motion. *HotSec* (2011).

[8] Olivier Cappé, Eric Moulines, and Tobias Rydén. 2006. *Inference in hidden Markov models*. Springer Science & Business Media.

[9] Z Berkay Celik, Earlence Fernandes, Eric Pauley, Gang Tan, and Patrick McDaniel. 2019. Program analysis of commodity IoT applications for security and privacy: Challenges and opportunities. *ACM Computing Surveys (CSUR)* (2019).

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* (2009).

[11] Qi Alfred Chen, Zhiyun Qian, and Z Morley Mao. 2014. Peeking into Your App without Actually Seeing It: {UI} State Inference and Novel Android Attacks. In *USENIX Security Symposium*.

[12] Yushi Cheng, Xiaoyu Ji, Wenyuan Xu, Hao Pan, Zhuangdi Zhu, Chuang-Wen You, Yi-Chao Chen, and Lili Qiu. 2019. MagAttack: Guessing Application Launching and Operation via Smartphone. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*.

[13] Myeongwon Choi, Sangeun Oh, Insu Kim, and Hyosu Kim. 2022. MagSnoop: listening to sounds induced by magnetic field fluctuations to infer mobile payment tokens. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*.

[14] Matteo Ciman and Katarzyna Wac. 2016. Individuals' stress assessment using human-smartphone interaction analysis. *IEEE Transactions on Affective Computing* (2016), 51–65.

[15] Federal Trade Commission. 2014. *Data brokers: A call for transparency and accountability*. Createspace Independent Pub, 1–101.

[16] Habiba Farrukh, Tinghan Yang, Hanwen Xu, Yuxuan Yin, He Wang, and Z Berkay Celik. 2021. S3: Side-Channel Attack on Stylus Pencil through Sensors. *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2021).

[17] Surjya Ghosh, Kaustubh Hiware, Niloy Ganguly, Bivas Mitra, and Pradipta De. 2019. Emotion detection from touch interactions during text entry on smartphones. *International Journal of Human-Computer Studies* (2019).

[18] Xiaoyi Gu, Leman Akoglu, and Alessandro Rinaldo. 2019. Statistical Analysis of Nearest Neighbor Methods for Anomaly Detection. In *Advances in Neural Information Processing Systems*.

[19] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*.

[20] Jianmeng Huang, Yan Xiong, Wenchao Huang, Chen Xu, and Fuyou Miao. 2019. SieveDroid: Intercepting Undesirable Private-Data Transmissions in Android Applications. *IEEE Systems Journal* (2019).

[21] ifixit. 2021. iPhone XS and XS Max Teardown. https://www.ifixit.com/Teardown/iPhone+XS+and+XS+Max+Teardown/113021.

[22] ifixit. 2021. Samsung S10 and S10e Teardown. https://www.ifixit.com/Teardown/Samsung+Galaxy+S10+and+S10e+Teardown/120331.

[23] Abdul Rehman Javed, Mirza Omer Beg, Muhammad Asim, Thar Baker, and Ali Hilal Al-Bayatti. 2020. AlphaLogger: Detecting motion-based side-channel attack using smartphone keystrokes. *Journal of Ambient Intelligence and Humanized Computing* (2020).

[24] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Advances in Neural Information Processing Systems* (2020).

[25] Konrad Kollnig, Anastasia Shuba, Max Van Kleek, Reuben Binns, and Nigel Shadbolt. 2022. Goodbye tracking? Impact of iOS app tracking transparency and privacy labels. *ACM FAccT* (2022).

[26] Moez Krichen. 2021. Anomalies detection through smartphone sensors: a review. *IEEE Sensors Journal* (2021).

[27] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *IEEE International Conference on Data Mining*.

[28] Yihao Liu, Kai Huang, Xingzhe Song, Boyuan Yang, and Wei Gao. 2020. MagHacker: eavesdropping on stylus pen writing via magnetic sensing from commodity mobile devices. In *International Conference on Mobile Systems, Applications, and Services*.

[29] Anindya Maiti, Oscar Armbruster, Murtuza Jadliwala, and Jibo He. 2016. Smartwatch-Based Keystroke Inference Attacks and Context-Aware Protection Mechanisms. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*.

[30] Seita Maruyama, Satohiro Wakabayashi, and Tatsuya Mori. 2019. Tap'n ghost: A compilation of novel attack techniques against smartphone touchscreens. In *IEEE Symposium on Security and Privacy (IEEE S&P)*.

[31] Nikolay Matyunin, Yujue Wang, Tolga Arul, Kristian Kullmann, Jakub Szefer, and Stefan Katzenbeisser. 2019. MagneticSpy: Exploiting Magnetometer in Mobile Devices for Website and Application Fingerprinting. In *ACM Workshop on Privacy in the Electronic Society*.

[32] James Clerk Maxwell. 1865. VIII. A dynamical theory of the electromagnetic field. *Philosophical transactions of the Royal Society of London* (1865).

[33] Kirk T McDonald. 2017. Magnetic field in a time-dependent capacitor. Joseph Henry Laboratories, Princeton University, Princeton.

[34] Maryam Mehrnezhad, Ehsan Toreini, Siamak F Shahandashti, and Feng Hao. 2016. Touchsignatures: identification of user touch actions and PINs based on mobile sensor data via javascript. *Journal of Information Security and Applications* (2016).

[35] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. 2012. Tapprints: Your Finger Taps Have Fingerprints. In *International Conference on Mobile Systems, Applications, and Services (MobiSys)*.

[36] Sashank Narain, Amirali Sanatinia, and Guevara Noubir. 2014. Single-Stroke Language-Agnostic Keylogging Using Stereo-Microphones and Domain Specific Machine Learning. In *ACM Conference on Security and Privacy in Wireless And Mobile Networks*.

[37] R. Ning, C. Wang, C. Xin, J. Li, and H. Wu. 2018. DeepMag: Sniffing Mobile Apps in Magnetic Field through Deep Convolutional Neural Networks. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*.

[38] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. 2012. Accessory: password inference using accelerometers on smartphones. In *Workshop on Mobile Computing Systems & Applications*.

[39] Muslum Ozgur Ozmen, Xuansong Li, Andrew Chu, Z Berkay Celik, Bardh Hoxha, and Xiangyu Zhang. 2022. Discovering IoT Physical Channel Vulnerabilities. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[40] Hao Pan, Lanqing Yang, Honglu Li, Chuang-Wen You, Xiaoyu Ji, Yi-Chao Chen, Zhenxian Hu, and Guangtao Xue. 2021. MagThief: Stealing private app usage data on mobile devices via built-in magnetometer. In *2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*.

[41] Beatrice Perez, Mirco Musolesi, and Gianluca Stringhini. 2019. Fatal attraction: identifying mobile devices through electromagnetic emissions. In *Conference on Security and Privacy in Wireless and Mobile Networks*.

[42] Giuseppe Petracca, Ahmad-Atamli Reineh, Yuqiong Sun, Jens Grossklags, and Trent Jaeger. 2017. AWare: Preventing Abuse of {Privacy-Sensitive} Sensors via Operation Bindings. In *USENIX Security Symposium*.

[43] Giuseppe Petracca, Yuqiong Sun, Ahmad-Atamli Reineh, Patrick McDaniel, Jens Grossklags, and Trent Jaeger. 2019. EnTrust: Regulating Sensor Access by Cooperating Programs via Delegation Graphs. In *USENIX Security Symposium*.

[44] John A Quinn and Masashi Sugiyama. 2014. A least-squares approach to anomaly detection in static and sequential data. *Pattern Recognition Letters* (2014).

[45] L. Rabiner and B. Juang. 1986. An introduction to hidden Markov models. *IEEE ASSP Magazine* (1986), 4–16.

[46] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 2019. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *USENIX Security Symposium*.

[47] C.E.T. Research. 2010. *CMOSET 2010 Microsystems and Sensors Track Presentation Slides*. CMOS Emerging Technologies. https://books.google.com/books?id=ekdkWGqw29EC

[48] I. W. Selesnick and C. S. Burrus. 1998. Generalized digital Butterworth filter design. *IEEE Transactions on Signal Processing* 46, 6 (1998), 1688–1694. https://doi.org/10.1109/78.678493

[49] Oleg Semenov, Hossein Sarbishaei, and Manoj Sachdev. 2008. *ESD Models and Test Methods*.

[50] Haoqi Shan, Boyi Zhang, Zihao Zhan, Dean Sullivan, Shuo Wang, and Yier Jin. 2022. Invisible Finger: Practical Electromagnetic Interference Attack on Touchscreen-based Electronic Devices. In *IEEE Symposium on Security and Privacy (IEEE S&P)*.

[51] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. 2019. A context-aware framework for detecting sensor-based threats on smart devices. *IEEE Transactions on Mobile Computing* (2019).

[52] Rui Song, Yubo Song, Shang Gao, Bin Xiao, and Aiqun Hu. 2018. I know what you type: Leaking user privacy via novel frequency-based side-channel attacks. In *IEEE Global Communications Conference (GLOBECOM)*.
[53] Raphael Spreitzer. 2014. Pin skimming: Exploiting the ambient-light sensor in mobile devices. In *ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. 51–62.
[54] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. 2017. Systematic classification of side-channel attacks: A case study for mobile devices. *IEEE Communications Surveys & Tutorials* (2017), 465–488.
[55] Statista. 2020. Apple's smartphone market share by sales to end users from first quarter 2016 to fourth quarter 2020. https://www.statista.com/statistics/1168529/global-apple-market-share-2020/.
[56] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems*, Vol. 27. 3104–3112.
[57] David MJ Tax and Robert PW Duin. 2004. Support vector data description. *Machine learning* (2004).
[58] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* (2017).
[59] Kai Wang, Richard Mitev, Chen Yan, Xiaoyu Ji, Ahmad-Reza Sadeghi, and Wenyuan Xu. 2022. GhostTouch: Targeted Attacks on Touchscreens without Physical Touch. In *USENIX Security Symposium*.
[60] Tete Xiao, Xiaolong Wang, Alexei A Efros, and Trevor Darrell. 2021. What Should Not Be Contrastive in Contrastive Learning. In *International Conference on Learning Representations*.
[61] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. 2010. A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter* (2010), 40–48.
[62] Zhi Xu, Kun Bai, and Sencun Zhu. 2012. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*.
[63] Tuo Yu, Haiming Jin, and Klara Nahrstedt. 2016. Writinghacker: audio based eavesdropping of handwriting via mobile devices. In *ACM International Joint Conference on Pervasive and Ubiquitous Computing*.
[64] Nan Zhang, Kan Yuan, Muhammad Naveed, Xiaoyong Zhou, and XiaoFeng Wang. 2015. Leave me alone: App-level protection against runtime information gathering on android. In *2015 IEEE Symposium on Security and Privacy*.

APPENDIX

A DETAILS OF BINARY TOUCH EXTRACTION MODEL

Figure 1 shows the structure of the CNN-LSTM network designed for extracting binary touch sequence from the 3-D magnetometer data. We used an initial development dataset to select the hyperparameters. The network consists of CNN layers for extracting features from the magnetometer data, followed by bidirectional LSTM layers for sequence-to-sequence mapping. We use two parallel CNN layers with filter sizes 5 and 10; each layer has 15 different filters. The features extracted from these two layers are concatenated and fed into a Bi-LSTM layer. Our network uses two bidirectional LSTM layers, each with size 50, for sequence-to-sequence mapping of the touch features. The LSTM layers are followed by a dropout layer with 30% dropout rate to avoid overfitting. The final layer is a dense layer with a softmax activation function. This layer outputs a binary prediction, 1 for touch and 0 for no-touch, for each sample.

B DETAILS OF EXPERIMENTS DESIGN

In this section, we show the details of the applications used in the different experiments. We divide the applications into 8 sets where each set includes the applications used together in our experiments by one participant at a time. Table 1 shows the open-source applications with the links of their source codes. It also shows the application set that each application belongs to. In Table 2, we show the details of the application sets used by each user.

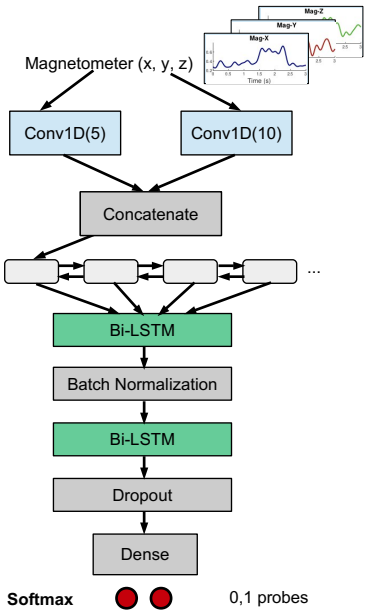


Figure 1: Binary touch extraction network.

C EFFECT OF TOUCH ON OTHER SENSORS

We have shown that touch events have no effect on the x-axis of the accelerometer and gyroscope signals when there is no significant motion in the phone. Figures 2 and 3 show the y-axis signal and the z-axis signal, respectively, of the gyroscope and accelerometer when a touch event occurs. Consistently, all the axes of these sensors are not affected by a touch event, unlike the magnetometer sensor, which does not depend on motion to detect touches.

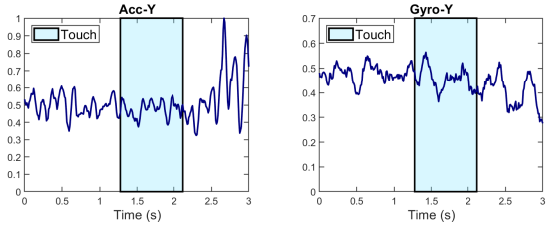


Figure 2: Showing that touch events have no effect on the y-axis signal of the accelerometer and gyroscope.

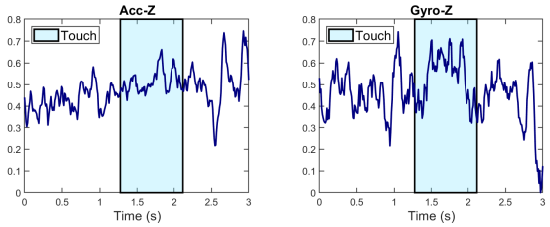


Figure 3: Showing that touch events have no effect on the z-axis signal of the accelerometer and gyroscope.

Table 1: Open source apps used in iStelan.

Apps Set	App Type	URL
Apps_A1	Board Game	https://github.com/austinzheng/swift-2048
	Browser	https://github.com/meismyles/SwiftWebVC
	Music	https://github.com/analogcode/Swift-Radio-Pro
	Chat	https://github.com/relatedcode/Messenger[3]
Apps_A2	Board Game	https://github.com/dave-abelson/CookieCrunch
	Browser	https://github.com/revblaze/WiBlaze
	Music	https://github.com/stellz/MusiCharts
	Chat	https://github.com/vitaliy-paliy/Messenger
Apps_A3	Board Game	https://github.com/ghewgill/puzzles
	Browser	https://github.com/amerhukic/Browser
	Music	https://github.com/Salmik/Apple-Music-Clone
	Chat	https://github.com/relatedcode/Messenger
Apps_B1	Maps	https://github.com/googlemaps/maps-sdk-for-ios-samples
	Shooting Game	https://github.com/woguan/Legend-Wings
	Shopping	https://github.com/openshopio/openshop.io-ios
Apps_B2	Maps	https://github.com/balitax/Google-Maps-Direction
	Shooting Game	https://github.com/FaustosWork/Seek-N-Destroy
	Shopping	https://github.com/shyamPindoria/Shopping-App
Apps_B3	Maps	https://github.com/balitax/Google-Maps-Direction [4]
	Shooting Game	https://github.com/r3econ/spacequest-ios
	Shopping	https://github.com/kaveenabeywansa/shoppingapp
Apps_C	ToDo	https://github.com/TarokhDev2020/FireTodo-for-iOS
	Finance Tracking	https://github.com/frozenstruct/iOS-FinanceApp
Apps_C'	Instagram	https://github.com/PankajGaikar/Instagram-Clone-SwiftUI
	Weather	https://github.com/jonstjohn/ClimbingWeather-iPhone
	Text Editor	https://github.com/JKKross/red$_z$ebra

Table 2: Details of users and app sets used during iStelan's experiments.

User	Apps_A1	Apps_A2	Apps_A3	Apps_B1	Apps_B2	Apps_B3	Apps_C	Apps_C'
1	✓						✓	
2	✓						✓	
3	✓	✓						
4	✓							
5	✓							
6	✓							
7	✓							
8	✓							
9				✓				
10				✓				
11				✓				
12				✓				
13				✓				
14				✓				
15				✓				
16				✓				
17				✓				
18			✓					
19					✓			
20						✓		
21								✓
22								✓

D COMPARISON AGAINST BASELINES

We perform a per-attack stage comparison against techniques of existing works. We additionally compare iStelan with recent side-channel attacks that detect touch events based on accelerometer and gyroscope motion sensors.

iStelan Binary Touch Extractor. We compare iStelan's binary touch extractor (`Stage-1`) with two other touch detection techniques. First, we implemented an adapted model that uses two LSTM layers to map the magnetometer signal to a binary touch signal, where LSTMs are commonly used for sequence mapping [56].

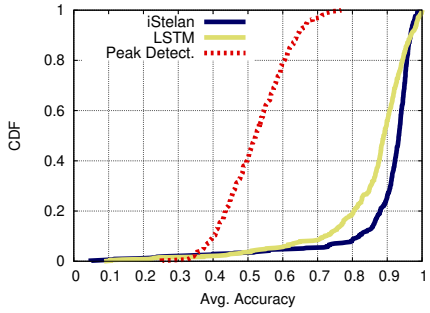


Figure 4: Comparison of iStelan binary touch extraction with baseline.

Second, we implemented a commonly used peak detection technique [5] on the continuous magnetometer signal that detects significant peaks of the signal that map to a touch event. Figure 4 shows the cumulative distribution function (CDF) of iStelan's accuracy compared to other techniques. We observe that iStelan's parallel CNN layers extracting features from the magnetometer signal improve the accuracy over the LSTM-based model and outperform the peak detection algorithm.

iStelan App Classifier. We implemented two baseline ML classifiers, Random forest (RF) and K-nearest neighbors (k-NN), to compare their results with iStelan's app classifier (`Stage-2`). While these classifiers do not directly capture the temporal aspects of the data, they could be used to classify sequence data using statistical features that summarize the data [61]. To train the classifiers, we extract statistical features from the binary touch data, including the histogram, average, maximum and minimum duration of touch events, and idle time. The average accuracy of the RF and k-NN classifiers is 63% and 56%. These classifiers give low accuracy as they fail to capture the temporal relationship between neighboring touch samples in a given window.

We also compare iStelan against a distance-based classifier that uses Dynamic Time Warping (DTW) as a similarity measure commonly used with time series data [61]. We apply this measure to the k-NN classifier (with $k = 3$). The average accuracy of this classifier is 40%. In contrast, iStelan achieves a higher accuracy of 74% using the LSTM model for classification. The LSTM model keeps track of the dependencies between the samples in a given sequence of binary touch data and dynamically captures data features, resulting in higher classification accuracy.

We also compare iStelan app classifier (`Stage-2`) to MagneticSpy [31], which uses the effect of electromagnetic emanation of the smartphone processor to detect the app being used. Since MagneticSpy's source code is not publicly available, we implement their app classifier and evaluate its accuracy on data collected from our experiments. MagneticSpy transforms the 3D magnetometer data into one-dimensional data using principle component analysis (PCA), extracts features from this data, and feeds it into a Random Forest classifier. For this comparison, we collect 30 samples of the 3D magnetometer data for 4 different apps. Each sample spans a 12 secs duration, which is the detection window size used in MagneticSpy. We follow the same experimental setup in [31], where a user opens each app for 12 secs and closes it without using the app while the phone is placed stationary on a table. Since some of the classifier parameters are not declared, we find the set of optimal parameters on our dataset. The accuracy of MagneticSpy for app detection is 24%, which is close to a random guess, while their reported results are up to 80%. This suggests that some external factors could bias their experiments. For example, the magnetic fingerprint used to detect apps could be affected by the ambient magnetic noise instead of the actual electromagnetic effect from the processor. This may also suggest that their technique does not work for iOS devices since all of their experiments for app detection were conducted on Android smartphones.

Table 3: Comparison with other motion sensors.

Sensor	Stage 1	Stage 2	Stage 3
Accelerometer	0.82	0.20	0.78
Gyroscope	0.90	0.27	0.78

iStelan Touch Type Detector. We compare iStelan touch type detector (`Stage-3`) to a baseline HMM model that uses fixed parameters. In our HMM model, we use continuous probability distributions for observation and transition probabilities, which depend on state duration and idle time between states (as detailed in Section 4.4). In the baseline model, we use fixed observation probability for each state that depends on the frequency of the particular state in the training dataset. Assuming the frequency of state `i` is $\mathtt{f_i}$ and the number of states in the training set is `f`, the observation probability is $\mathtt{f_i/f}$. Similarly, the transition probability from state `i` to `j` is $\mathtt{f_{i,j}/f_i}$, where $\mathtt{f_{i,j}}$ is the frequency of transitions from `i` to `j`. This baseline model yields 67% accuracy, lower than 73% iStelan accuracy since our HMM model not only captures the frequency of the state transitions but also the duration between state transitions allowing iStelan to differentiate between various touch types.

We also compare using one HMM for all app types data against using a separate model for each app type. The result for using one HMM model is 51% compared to 73% with separate models. This is because the frequency of touch types, which is used to compute the model probabilities, differs for each app type.

Detection Touch Events with Motion Sensors. We compare iStelan with other side-channel attacks that detect touch events based on accelerometer and gyroscope sensors. We collected motion sensor data from 5 users while using the 4 different apps with the phone lying on a table. We use this data to evaluate the three attack stages of iStelan. We split the data into 4 users for training and 1 user for testing. The results are shown in Table 3. The accelerometer data failed to detect the binary touches and outputs only zero (which was the majority class 82%). While the gyroscope gives 90% for binary detection, it yields a low accuracy close to random for the app classification, which shows that it also outputs a majority of zeros in binary. Both sensors output one class of "taps", which was 78% of the data. This shows that these sensors fail to detect the side-channel when there is no significant motion in the phone.

[3] An old version of the app from 2019 with a different UI.
[4] For the maps app, we manually edited the app from set Apps_B2 to change the UI

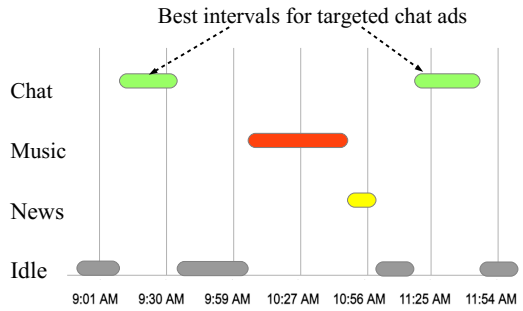


Figure 5: An example of victim's apps usage monitored by a data broker app using iStelan system.

E CASE STUDIES

We discuss two case studies to show the practical impact of iStelan side-channel attack.

E.1 GUI Confusion Attacks

In this case study, we show how iStelan can be used to launch GUI-confusion attacks [3], which stealthily prompt the user to enter her sensitive information. In this attack scenario, the malicious app first infers the app type of a user with iStelan, and launches phishing and click-jacking attacks by mimicking the GUI of other apps.

The iOS does not allow a background app to show a pop-up screen; however, we show that the adversary can launch a GUI-confusion attack in three different ways. In the *active app switch*, the attack occurs while the user is actively using the target app (See Figure 6a). Instead of showing a pop-up screen, the malicious app sends a push notification at the top of the screen. This notification contains a message that superstitiously appears to the user as a system notification or originating from the target app. Although the iOS displays the app name and icon on top of the notification, the malicious app uses a generic name, less noticed by the user. The notification also allows adding an image on the right side. In Figure 6a, we added the target app icon, which could further lead to user confusion. Furthermore, iOS allows apps to design a custom notification, including a button that prompts the user to click. If the user clicks on the notification, an app switch occurs to the malicious app. The malicious app displays a view that mimics the target app with a prompt to enter sensitive information such as login credentials or credit card information.

The *passive app switch* attack, as shown in Figure 6b, occurs when the malicious app detects the target app type (after being used by the user for a sufficient time interval), changes its UI to be similar to the target app in the background, and waits in idle state. The malicious app stays idle in the background. While the user is switching between apps looking for the target app, she could inadvertently switch to the malicious app.

Lastly, in *full-screen attack* passive attack, when the user uses the malicious app, the app goes to the full-screen mode and creates a fake home screen with standard actions. This gives the user the impression that she interacts with the OS while the malicious app receives her inputs.

In the previous attacks, we assume that an adversary correctly guesses the exact app the user is using. Yet, even with only knowing the app type, the attacker can conduct GUI-confusion attacks. For example, if iStelan identifies the user uses a chatting app, the malicious app could send a notification with a one-week free international calls that requires entering payment information. If the user clicks the notification to subscribe, an app switch occurs, and the malicious app shows a generic payment web view that would be perceived by the user as if originating from the target app.

(a) Active switch **(b) Passive switch**

Figure 6: Examples of GUI confusion attacks with Messenger as target app and the malicious app named "ALT".

E.2 Targeted Ads Attacks

We show another case study where the data inferred by iStelan could also be valuable for companies aiming to advertise their web or mobile apps to interested users. These companies usually buy user data from data brokers, where they can analyze the behavior of users and target ads to specific categories of users at the best time intervals [15].

We construct an attack in which an attacker leverages iStelan to act as a data broker. While the victim user naturally uses her phone during a time frame, the data broker app collects magnetic sensor data in the background. iStelan infers the type of apps a user has used during this time frame and constructs a timeline of apps usage along with idle time. Figure 5 illustrates an example of the victim's apps usage timeline as constructed by iStelan during a three hours interval. Assuming a web app or another app installed on the victim's device sends their new chatting app ads. Therefore, they buy this timeline data from the data broker to extract information about which users use chatting apps the most and at which time of the day to target the ads.