# Pitfalls in Designing Zero-Effort Deauthentication: Opportunistic Human Observation Attacks

Otto Huhta*, Prakash Shrestha†, Swapnil Udar*, Mika Juuti*, Nitesh Saxena† and N. Asokan‡
*Aalto University
†University of Alabama at Birmingham
‡Aalto University and University of Helsinki
{otto.huhta, swapnil.udar, mika.juuti}@aalto.fi, {saxena, prakashs}@uab.edu, asokan@acm.org

***Abstract*—Deauthentication is an important component of any authentication system. The widespread use of computing devices in daily life has underscored the need for *zero-effort* deauthentication schemes. However, the quest for eliminating user effort may lead to hidden security flaws in the authentication schemes.**

**As a case in point, we investigate a prominent zero-effort deauthentication scheme, called ZEBRA, which provides an interesting and a useful solution to a difficult problem as demonstrated in the original paper. We identify a subtle incorrect assumption in its adversary model that leads to a fundamental design flaw. We exploit this to break the scheme with a class of attacks that are much easier for a human to perform in a realistic adversary model, compared to the naïve attacks studied in the ZEBRA paper. For example, one of our main attacks, where the human attacker has to opportunistically mimic only the victim's keyboard typing activity at a nearby terminal, is significantly more successful compared to the naïve attack that requires mimicking keyboard and mouse activities as well as keyboard-mouse movements. Further, by understanding the design flaws in ZEBRA as cases of *tainted input*, we show that we can draw on well-understood design principles to improve ZEBRA's security.**

## I. Introduction

User authentication is critical to many on-line and off-line services. Computing devices of all types and sizes, ranging from mobile phones through personal computers to remote servers rely on user authentication. *Deauthentication* – promptly recognizing when to terminate a previously authenticated user session – is an essential component of an authentication system.

The pervasive use of computing in people's daily lives underscores the need to design effective, yet intuitive and easy-to-use deauthentication mechanisms. However, this remains an important unsolved problem in information security. A promising approach to improving usability of (de)authentication mechanisms is to make them *transparent* to users by reducing, if not eliminating, the cognitive effort required from users. Although such *zero-effort* authentication schemes are compelling, designing them correctly is difficult. The need to minimize additional user interactions required by the scheme is a severe constraint that can lead to design decisions which might affect the security of the scheme.

One prominent approach for improving usability of security mechanisms involves comparing information observed from two different sources. Such a *bilateral* approach has been proposed as part of solutions for a variety of security problems such as deauthentication of users [23], determining if two or more devices are co-present in the same place [28], establishing security associations among nearby devices ("pairing") [29], [9] and authorizing transactions between co-present devices [8]. Bilateral authentication schemes are attractive because they can avoid imposing any cognitive load on users (thus making them "zero-effort"), or the need to store sensitive or user-specific information on devices [23]. However, an adversary capable of influencing one or both sources of information being compared in a bilateral scheme may compromise security.

In this paper, we illustrate the problem of subtle flaws in the design of zero-effort bilateral schemes by examining an interesting class of schemes represented by ZEBRA, a zero-effort bilateral deauthentication scheme, proposed recently in a premier security research venue [23]. ZEBRA is intended for scenarios where users authenticate to "terminals" (such as desktop computers). In such scenarios, users typically have to either manually deauthenticate themselves by logging out or locking the terminal, or the terminal can deauthenticate a user automatically after a sufficiently long period of inactivity. The former requires user effort while the latter sacrifices promptness. ZEBRA attempts to make the process of deauthentication *both prompt and transparent*: once a user is authenticated to a terminal (using say a password), it continuously, yet transparently *re-authenticates* the user so that prompt deauthentication is possible without explicit user action. A user is required to wear a bracelet equipped with sensors on his mouse-holding hand. The bracelet is wirelessly connected to the terminal, which compares the sequence of events it observes (e.g., keyboard/mouse interactions) with the sequence of events inferred using measurements from the bracelet sensors. The logged-in user is deauthenticated when


NDSS '16, 21-24 February 2016, San Diego, CA, USA

http://dx.doi.org/10.14722/ndss.2016.23199

the two sequences no longer match.

ZEBRA is particularly compelling because of its simplicity of design. However, the simplicity hides a design assumption that an adversary can exploit to defeat the scheme. We show how a more realistic adversary can circumvent ZEBRA. Since no implementation of ZEBRA was available, we built an end-to-end implementation and use it in our attack. We also implemented changes needed to make ZEBRA work in real-time.

Our primary contributions can be summarized as follows:

1) We highlight fundamental pitfalls in designing zero-effort bilateral security schemes by studying ZEBRA, a notable prior scheme. We identify a hidden design choice in ZEBRA that allows us to develop an **effective attack strategy**: a human attacker observing a victim at a nearby terminal and *opportunistically* mimicking only a subset of the victim's activities (e.g., keyboard events) at the authentication terminal (Section III).
2) We build a **end-to-end implementation**[1] of ZEBRA (Section IV), and demonstrate via **experiments in realistic adversarial settings** that ZEBRA as designed can be defeated by our opportunistic attacker with a (statistically) significantly higher probability compared to a naïve attacker, also considered in [23] (one who attempts to mimic all, keyboard and mouse, activities) (Section V).
3) We cast ZEBRA's design flaw as a case of **tainted input**, and thus draw from well-understood principles of secure system design that may help improve the security of ZEBRA (Section VI).

## II. Background

Since we use ZEBRA [23] as our exemplary bilateral zero-effort deauthentication scheme, we now describe it in more detail. It is intended for multi-terminal environments where users frequently move between terminals. Mare et al. [23] present a hospital environment as their motivating scenario. Hospital staff members often use shared terminals. However, a user must not, intentionally or unintentionally, access hospital systems from terminals where other users have logged in. Users may leave terminals without logging out, but may still remain in the vicinity. Proximity-based zero-effort deauthentication schemes such as ZIA [12] or BlueProximity [5] cannot be used because these methods are not accurate enough for short distances. Although the motivating scenario is an environment with shared terminals, zero-effort deauthentication schemes like ZEBRA are broadly applicable to any scenario where users may leave their terminals unattended.

**Adversary Model**: ZEBRA[23] considers two types of adversaries: "innocent" and "malicious". An innocent adversary is a legitimate user who starts using an unattended terminal inadvertently without realizing that another user ("victim") is logged into that terminal. In contrast, a malicious adversary deliberately uses an unattended terminal of the victim with the intent of performing some action impersonating the victim. A malicious adversary may observe the behavior and actions of the victim (such as imitating the victim's hand movements made while interacting with another terminal). The goal of ZEBRA is to quickly detect if a previously authenticated session on a terminal is being used by anyone other than the user who originally authenticated, and promptly deauthenticate the session. Naturally, decisions made by ZEBRA should minimize false positives (incorrectly recognizing an adversary as the original authenticated user, thereby failing to deauthenticate him as well as false negatives (incorrectly concluding that current user is not the original user, thereby deauthenticating him.

**System Architecture**: Figure 1 depicts the normal (benign) operation of ZEBRA. It correlates a user's activities on a terminal with measurements of user activity relayed from a wrist-worn device (we call it a bracelet for simplicity, but it can be a general-purpose smartwatch as in our implementation and analysis). The goal is to continuously verify that the logged in user is the one using the terminal and to quickly deauthenticate any unintended users. ZEBRA assumes terminals with keyboard/mouse and a personal bracelet for each user of the system. The bracelet has accelerometer and gyroscope sensors to record wrist movements. Terminals and bracelets securely communicate using "paired" wireless channels like Bluetooth. In addition, a terminal knows the identity of the bracelet associated with each authorized user. Users initially authenticate themselves to terminals using some mechanism external to ZEBRA (such as using a username/password). Once a user has been authenticated, the terminal connects to that user's bracelet and starts receiving sensor measurements from it.

The basic principle of operation is to compare the sequence of user activity seen at the terminal with that inferred from data sent by the bracelet. ZEBRA's system architecture is shown in Figure 2. An *Interaction Extractor* on the terminal identifies the *actual interaction sequence* based on input events observed by the terminal peripherals. It defines three different types of such interactions: typing, scrolling, and hand movements between the mouse and keyboard (referred to as "MKKM")[2]. Interaction Extractor records the timestamps of each event in the actual interaction sequence. A *Segmenter* on the terminal receives measurement data sent by the bracelet and segments this data according to the timestamps it receives from Interaction Extractor. Segmenter ignores all measurements that fall outside these time slots. From the segments, a *Feature Extractor* extracts salient features and feeds them to an *Interaction Classifier* that has been trained to identify the type of interaction from bracelet measurement data. The classifier outputs a *predicted interaction sequence*. Finally,

[1] Unlike [23] which only described the implementation of individual components and off-line classification.

[2] ZEBRA neither cares about which key was pressed nor about what direction the mouse was scrolled. It actually cares about whether the interaction is typing, scrolling or movement between mouse and keyboard.

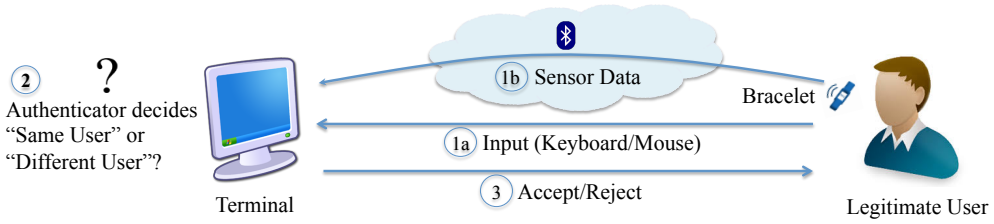



Fig. 1: Normal operation of ZEBRA

an *Authenticator* compares the two interaction sequences and determines whether the current user at the terminal is the "same" as, or "different" from, the originally authenticated user.

Authenticator can be tuned by a number of parameters. It compares sequences of length *w (window size)* at a time. In each window, if the fraction of matching interactions exceeds a threshold *m (matching threshold)*, it records 1 for that window; otherwise it records 0. If the record is 0 for *g (grace period)* successive windows, the authenticator outputs "different" causing ZEBRA to deauthenticate the session. Successive windows may overlap, as determined by *f (overlap fraction)*, with 0 signifying no overlap.

Segmenter ignores readings from the bracelet when Interaction Extractor detects no activity on the terminal. The choice was motivated in [23] by privacy considerations: the user's activities are not monitored when nobody is using the terminal. At first glance, it is a natural and reasonable design decision: if there is no terminal activity, there is reason to deauthenticate the session (thus reducing the chances of false negative decisions). However, as we shall see, an adversary can exploit this subtle aspect of the design.

**Validation**: Mare et al. [23] validated usability of their deauthentication scheme by calculating false negative rates for normal usage scenarios with different parameter settings. They validated the security by considering three separate scenarios. The first two scenarios model the "innocent adversary": the logged in user (victim) is either walking or writing nearby while the attacker accesses the victim's terminal. The last scenario models the "malicious" adversary: the victim uses *another* terminal, while the attacker uses the victim's original terminal. The activity conducted by both victims and attackers is filling forms. These scenarios were chosen as representative of multi-user environments such as hospitals, where physicians enter form-type data about their patients and routinely forget to log out of their terminals. It is reasonable to assume there are multiple terminals that users access and use. Similar usage scenarios are plausible in other contexts as well, such as in factory floors or control rooms. In [23], the malicious adversary is required to mimic *all* mouse-hand movements of the victim. Ordinary non-expert users act as the attackers in their analysis. Because Mare et al. [23] "realize that a real adversary can be motivated and skilled enough to mimic user very well, compared to our adversaries", they tried to make the scenario advantageous to the attacker by (a) providing the attacker with a clear view of the victim's screen and (b) have the victim give verbal cues to indicate what the victim was doing during the experiments (e.g., answering question 2 in the form). They concluded that their system was able to deauthenticate such attackers in reasonable time, while keeping false negative rates low.

## III. Our Attack

There are a number of attributes that make ZEBRA attractive. In particular, rather than trying to *recognize* the user, ZEBRA's bilateral approach simply *compares* two sequences that characterize user interaction. Consequently, its decisions neither limit how a user interacts with the terminal nor require storing any information about the user or his style of interaction. Such simplicity makes ZEBRA robust but also vulnerable. In this section, we revisit the security analysis in [23], point out a design flaw, and explain how it can be used to attack ZEBRA.

### A. Revisiting ZEBRA Security Analysis

Recall from Section II that Segmenter ignores all measurement data from the bracelet during periods when Interaction Extractor does not record any activity on the terminal involving the three types of interactions recognized by ZEBRA. However, the attacked terminal is under the control of the adversary and thus she can effectively choose which parts of the bracelet measurement data will be used by ZEBRA to re-authenticate the user. Mimicking all interactions is not the best attack strategy. A smart adversary can selectively choose only a subset of the victim's interactions to mimic since it can ensure that the rest of the victim's interactions will be ignored by Authenticator. Furthermore, to validate security, we need to use a realistic adversary model which allows attackers to be skilled and experienced in mimicking how people interact with terminals. It is unreasonable to use inexperienced test participants to model the adversary. Thus, the role of the attacker in this paper was played by two members of our research group that were knowledgeable of the ZEBRA system and experienced at mimicking attacks.

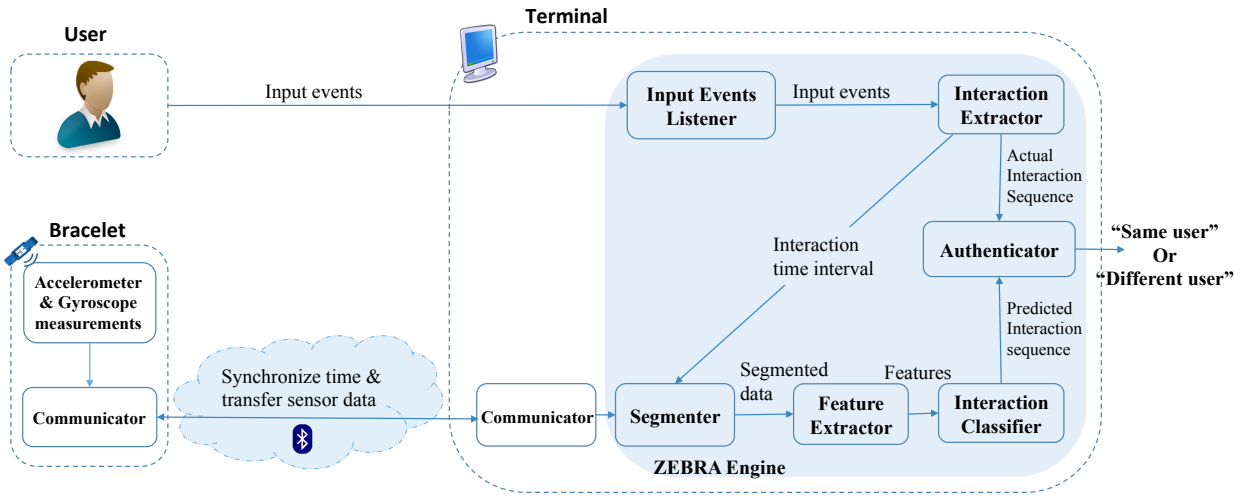



Fig. 2: ZEBRA system architecture

### B. Attack Scenarios and Strategies

In our attack scenarios, we model a malicious adversary against ZEBRA as discussed in Section II. We assume that the adversary $\mathcal{A}$ accesses the attacked terminal $\mathcal{AT}$ when the victim $\mathcal{V}$ steps away from it without logging out. We also assume that $\mathcal{V}$ is using another computing device (the "victim device", $\mathcal{VD}$) elsewhere (e.g., a nearby terminal). Figure 3 illustrates the attack setting.

**Strategy**: The goal of $\mathcal{A}$ is to remain logged in on $\mathcal{AT}$ for as long as possible, while interacting with the terminal. To this end, $\mathcal{A}$ needs to consistently produce a sufficiently large fraction of interactions that will match $\mathcal{V}$'s interactions on $\mathcal{VD}$. Since $\mathcal{AT}$ is under the control of $\mathcal{A}$, it can choose when $\mathcal{AT}$'s Interaction Extractor triggers Authenticator to compare the predicted and actual interaction sequences. If $\mathcal{A}$ adopts an *opportunistic* strategy, it can *selectively* choose only a subset of $\mathcal{V}$'s interactions to mimic so as to maximize the fraction of matching interactions. We conjecture that such an opportunistic adversary will be more successful than the naïve adversary that was considered in [23].

First, we consider a *keyboard-only* attack where $\mathcal{A}$ mimics only the typing interactions while ignoring all others. Typing sequences are typically longer and less prone to delays in mimicking. The opportunistic strategy is for $\mathcal{A}$ to start typing only after $\mathcal{V}$ starts typing and attempt to stop as soon as $\mathcal{V}$ stops. A sophisticated keyboard-only attacker may estimate the expected length of $\mathcal{V}$'s typing session and attempt to stop before $\mathcal{V}$ does. If $\mathcal{A}$ makes just a few key presses each time $\mathcal{V}$ begins typing, he can be confident that the actual interaction sequence he produces will match the predicted interaction sequence. These keyboard-only attacks are powerful because in all modern personal computer operating systems a wide range of actions can be performed using only the keyboard.

Second, we consider an *all-activity* attack, where $\mathcal{A}$ mimics all types of interactions (typing, scrolling and MKKM) but opportunistically chooses a subset of the set of interactions. As before, the $\mathcal{A}$'s selection criterion is the likelihood of correctly mimicking $\mathcal{V}$. In particular, $\mathcal{A}$ will use the following strategy:

- Once $\mathcal{A}$ successfully mimics a keyboard to mouse interaction, he is free to carry out any interaction involving the mouse (scroll, drag, move) at will because the bracelet measurements for all interactions involving the mouse are likely to be similar.
- If $\mathcal{A}$ fails to quickly mimic a keyboard to mouse (or vice versa) interaction, he does nothing until the next opportunity for an MKKM interaction arises (foregoing all interactions until after the MKKM is completed).

ZEBRA concatenates continual typing events into up-to 1 second long interactions: as such the typing speed of $\mathcal{A}$ is not particularly relevant. Instead, $\mathcal{A}$ may divert more of his attention to observing $\mathcal{V}$.

**Observation Channels**: By default, and similar to [23], we consider an adversary $\mathcal{A}$ who has a clear view of $\mathcal{V}$'s interactions (Figure 3). This models two cases: where $\mathcal{A}$ has direct visual access to $\mathcal{V}$ and where $\mathcal{A}$ has access to a *video aid* such as a surveillance camera aimed at $\mathcal{VD}$. During our attacks that use visual information of the victim's behavior, victim's new device $\mathcal{VD}$ was placed next to the victim terminal $\mathcal{AT}$. We also consider the case where $\mathcal{A}$ has no visual access to but can still hear sounds resulting from $\mathcal{V}$'s activities. Again, this models two cases: where both $\mathcal{V}$ and $\mathcal{A}$ are in the same physical space separated by a visual barrier (e.g., adjacent cubicles) and where $\mathcal{A}$ has planted an *audio aid* (e.g., a small hidden bug or a microphone) close to $\mathcal{VD}$.

**Scenarios**: The combination of attack strategy and type of observation channel leads to several different attack scenarios. We consider four of the most significant ones:

- In **naïve all-activity** attack, $\mathcal{A}$ is able to both see and hear $\mathcal{V}$. $\mathcal{A}$ attempts to mimic *all interactions* of $\mathcal{V}$. This is the attack scenario proposed and studied in [23].
- In **opportunistic keyboard-only** attack, $\mathcal{A}$ is able to both see and hear $\mathcal{V}$. $\mathcal{A}$ selectively mimics only a *subset of $\mathcal{V}$'s typing interactions*.
- In **opportunistic all-activity** attack, $\mathcal{A}$ is able to both see and hear $\mathcal{V}$. $\mathcal{A}$ selectively mimics a *subset of all types of interactions* of $\mathcal{V}$ following the guidelines mentioned

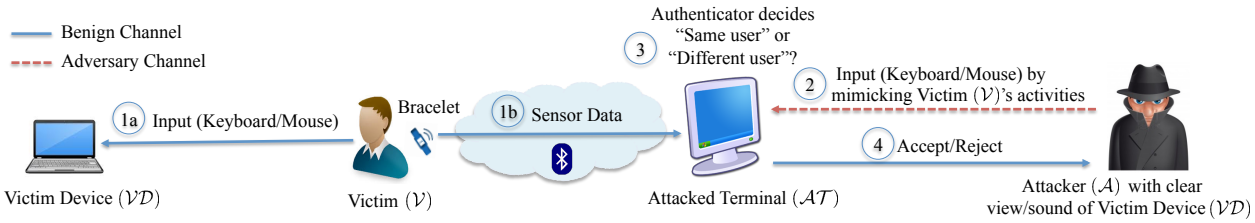



Fig. 3: Basic attack setting

above.

- In **audio-only opportunistic keyboard-only** attack, $\mathcal{A}$ is able to hear, but not see, $\mathcal{V}$'s interactions. $\mathcal{A}$ listens for keyboard activity and attempts to mimic *a subset of $\mathcal{V}$'s typing interactions*.

While one can imagine other attack combinations, we consider these four to be representative of different choices available to $\mathcal{A}$. For example, we leave out an audio-only all-activity attack because it is unlikely to succeed. Although our experiments are "unaided" (i.e., no audio or video recording), the results generalize to aided scenarios, if data transmission between the aid and the attacker does not introduce excessive delays.

## IV. ZEBRA End-to-End System

Mare et al [23] describe a framework for ZEBRA and implemented some individual pieces. However, this was not a complete system. Therefore, we needed to build an end-to-end system from scratch to evaluate our conjecture about opportunistic attacks. Our goal was to make this system as close to the one in [23] as possible. We now describe our system and how we evaluated its performance.

### A. Design and Implementation

**Software and Hardware**: We followed the ZEBRA system architecture as described in Figure 2. Our system consists of two applications: the bracelet runs an Android Wear application and the terminal runs a Java application. Interaction Classifier is implemented in Matlab. Communicator modules in both applications orchestrate communication over Bluetooth to synchronize clocks between them and to transfer bracelet measurements to the terminal. The rest of the terminal software consists of the "ZEBRA Engine" (shaded rectangle) with the functionality described in Section II. The bracelet and terminal synchronize their clocks during connection setup. For our experiments, we used a widely available smartwatch (4GB LG G Watch R with a 1.2 GHz CPU and 512MB RAM) with accelerometer/gyroscope as the bracelet and standard PCs as terminals.

**Parameter Choices**: Mare et al [23] do not fully describe the parameters used in their implementation of ZEBRA components. Wherever available, we used the exact parameters provided in [23] [22]. For the rest, we strived to choose reasonable values. A full list of parameters and rationales for choosing their values appears in Appendix A.

**Classifier**: We use the Random Forest [7] classifier. Again, as [23] did not include all details on how their classifier was trained and tuned, we made parameter choices that gave the best results. Our forest consisted of 100 weak-learners. Each split in a tree considered $sqrt(n)$ features, where $n = 24$ was the total number of features, and the trees were allowed to fully grow. In addition, classes were weighted to account for any imbalances in the training dataset (described below in Section IV-B). We adopt the same set of features used in [23], and extract them for both accelerometer and gyroscope segments. A full list appears in Appendix **??**.

**Differences**: Despite our efforts to keep our system similar to that in [23], there are some differences. First, we wanted to use commercially widely available general-purpose smartwatches as bracelets. They tend to be less well-equipped compared to the high-end Shimmer Research bracelet used in [23]. Our smartwatch has a maximum sampling rate of around 200 Hz, whereas the Shimmer bracelet had a sampling frequency of 500 Hz. We discuss the implications of this difference in Section VII.

In addition, [23] mentions a rate of 21 interactions in a 6s period (3.5 interactions per second). However, in our measurements, users filling standard web forms averaged around 1.5 interactions per second. Their typing interactions were slightly less than 1s long on average and MKKM interactions typically spanned 1-1.5s. With our chosen parameters we could produce a rate of 3.5 interactions per second only in sessions involving hectic activity – switching extremely rapidly between a few key presses and mouse scrolls. Such a high rate could not be sustained in realistic PC usage.

### B. Data Collection

In our study, we recruited 20 participants to serve as users (victims) of the system. They were mostly students recruited by word of mouth (ages 20–35, 15 males; 5 females, all right-handed). Participation was voluntary, based on explicit consent. The study included both dexterous typists and less-experienced ones. Initially, we told the participants that the purpose of the study was to collect information on how they typically use a PC. At the end of the study, we explained the actual nature of the experiment. The members of our research groups played the role of the adversary $\mathcal{A}$, compared to the untrained users in [23]. No feedback was given to $\mathcal{A}$ whether a given attack attempt was successful or not.

Experiments were conducted in a realistic office setting (with several other people working at other nearby desks).

During a session, a participant did four 10-minute tasks filling a web form, in a similar setting as in [23]. From each task, two sets of user data were collected simultaneously: accelerometer and gyroscope measurements from the user's bracelet and the actual interaction sequence extracted by Interaction Extractor on the terminal. An attacker $\mathcal{A}$ assigned to a participant $\mathcal{V}$ conducted each of the four types of attack scenarios from III-B in turn. In the first three scenarios, $\mathcal{A}$ had direct visual access to $\mathcal{V}$. In the fourth scenario, we placed a narrow shoulder-high partition between $\mathcal{V}$ and $\mathcal{A}$ so that $\mathcal{A}$ can hear but not see $\mathcal{V}$. The 20 sessions thus resulted in a total of 80 samples, with each sample consisting of three traces: bracelet data of the user, actual interaction sequence of the user, and the actual interaction sequence of the attacker. All traces within a sample were synchronized. No other information (e.g., the content of what the participant typed in) was recorded. Participants were told what data was collected.

The data collection and the study followed IRB procedures at our institutions. The data we collected has very little personal information. It is conceivable that the interaction sequences or bracelet data could potentially be used to link a participant in our study to similar data from the same participant elsewhere. For this reason, we cannot make our datasets public, but will make them available to other researchers for research use.

### C. Performance Evaluation

**Usability:** To evaluate usability, we follow the same approach as in [23] to compute the false negative rate (FNR) as the fraction of windows in which Authenticator comparing the actual and predicted interaction sequences from the same user incorrectly outputs "different user." We employ the leave-one-user-out cross-validation approach: for each session, we train a random forest classifier using the 76 samples of bracelet data from all the other 19 sessions. We then use the four samples from the current session to test the classifier. We thus train 20 different classifiers, and report results aggregating classification of 80 samples in all.

Figure 4a shows how different window size ($w$) and matching threshold ($m$) values affect average FNR. As can be seen, FNR is very low for our system. The original ZEBRA paper [23] reports FNRs in the range of 0-16% whereas in our system the FNRs are 0-6%, and below 1% for window sizes above 10.

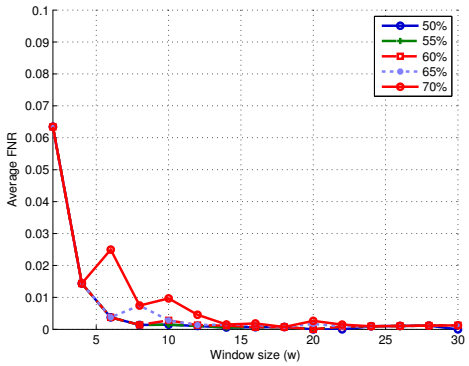
We also estimated the length of time (in terms of the number of windows) for which a legitimate user remained logged in. For this, we fix $w = 20$ and $m = 60\%$ as in [23]. On average, a window was 13 seconds long. The low FNRs result in no legitimate users getting logged out in any of the 10 minute samples. Figure 4b depicts this by plotting the fraction of users still logged in after a given number of authentication windows. The situation is the same when allowing one additional failed authentication window before logging a user out ($g = 2$), or when directly logging the user out after the first failed window ($g = 1$). This also seems in line with the results reported in [23], where one legitimate user was logged out when using a stricter grace period ($g = 1$).

Table I presents the confusion matrix for the classification performance of our Interaction Classifier. It combines data for all 80 (20 x 4) classifications. It shows that our system is very good at recognizing events accurately. For example, for the *typing* events, we obtain a precision of 96.9% (15753/16252) and a recall of 96.5% (15753/16332).

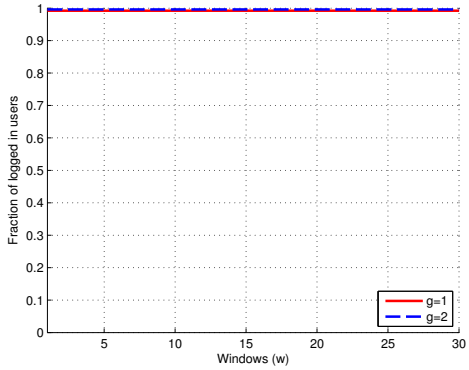
TABLE I: Confusion matrix for 80 legitimate user samples.

| | | **Predicted** | | |
|---|---|---|---|---|
| | | **Typing** | **Scrolling** | **MKKM** |
| **Actual** | **Typing** | 15753 | 354 | 225 |
| | **Scrolling** | 271 | 2506 | 2 |
| | **MKKM** | 228 | 71 | 15378 |

**Detection of Innocent Adversaries**: To estimate the security against an innocent adversary (a different user) who inadvertently starts using an unattended terminal where another user has logged in, we compute the true negative rate (TNR) for "mismatching" sequences: where the actual interaction sequence of one sample is compared against the predicted

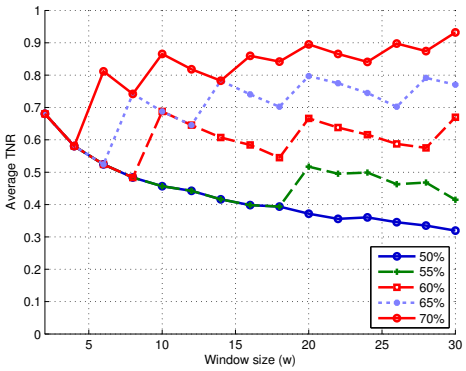



(a) Average FNR vs. window size ($w$) for different threshold ($m$) values. Fraction of windows that are incorrectly classified as mismatching.

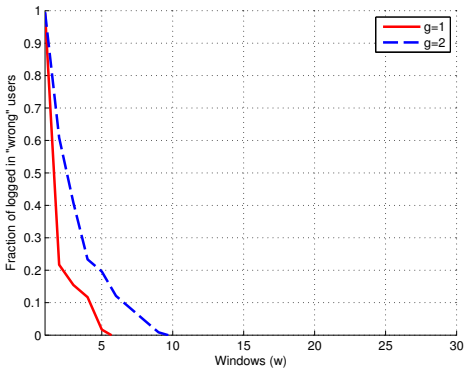



(b) Fraction of users remaining logged in after ($n$) authentication windows (with $w = 20, m = 60\%$), for different grace periods ($g$).

Fig. 4: Performance for legitimate users




(a) Average TNR for different threshold ($m$) values. Fractions of windows that correctly identify a wrong user.




(b) Fraction of users remaining logged in after ($n$) authentication windows (with $w = 20, m = 60\%$), for different grace periods ($g$).

Fig. 5: Performance for "wrong" (mismatched) users. Simulated accidental usage of the terminal.

interaction sequence of a *different sample*. With such mismatched sequences, the TNR is the fraction of windows in which the "wrong" user is correctly classified as "different user." Recall that data within a sample (and thus the interaction sequences extracted from it) are synchronized. When mismatching samples to compute TNR, we synchronized traces by aligning the starting points of the sequences being compared.

Figure 5a shows how different $w$ and $m$ values impact the average TNR (over 20 x 4 classifications) of our system with mismatched traces as input. Especially for thresholds of 60-70%, a majority of the authentication windows are identified correctly as non-matching. Again, using $w = 20$ and $m = 60\%$, Figure 5b shows the fraction of "wrong" users who remain logged in (i.e., incorrectly *not* deauthenticated) after interacting with the terminal for a given number of windows.

When the legitimate user is also interacting with a terminal, it can be expected that a non-zero fraction of actual interactions by the "wrong user" will accidentally match the predicted interactions by the legitimate user. As such ZEBRA Authenticator will accept (output 1) for a fraction of authentication windows. However, as can be seen from the fraction of logged in users in Figure 5b, a majority of users will quickly get logged out as any such accidental matches are not sufficient to keep the user logged in for an extended period of time. Using a strict grace period ($g = 1$), 78% of wrong users are logged out after the first authentication window and all but one after 5 windows. For $g = 2$, 80% of wrong users are logged out after 5 windows, and all by window 10.

To further evaluate the resilience of our system, we investigated the impact on FNR if the predicted and actual interaction sequences are desynchronized. We shifted the actual interaction sequence in each sample forward in time to simulate the time delay incurred, for example, when an attacker mimics his victim. Delays of 200 ms increase the Negative Rates (NR) from the 0-6% (presented in Figure 4a) to 1-20%, resulting in 5-10% of legitimate users getting logged out. Further increasing the delay to 500 ms increases the NR to 25-70% causing a majority of users to be logged out within 2-4 authentication windows. Thus, despite its low FNRs for legitimate users, our system is robust because it is sensitive to delays introduced in mimicking user interactions.

**Summary**: We therefore conclude that our end-to-end system is functionally comparable to that of [23]. Legitimate users remain logged-in at a very high rate, whereas the majority of wrong users are quickly logged out. Our system achieves lower FNR for legitimate users compared to [23], which is good for usability but may also be caused if the system is too permissive. However, our experiments with mismatched and desynchronized traces show marked increases in FNR suggesting that our system is not overly permissive.

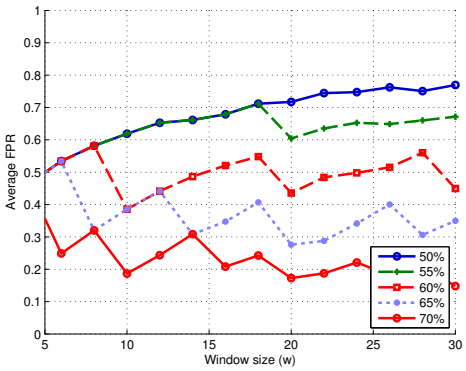
## V. Malicious Adversaries

Having shown that our end-to-end system is resilient against innocent adversaries, we now consider its security against malicious adversaries who attempt to intentionally mimic a victim's interactions. We consider the four types of attack scenarios from Section III-B: naïve and opportunistic all-activity attacks, and two variants of opportunistic keyboard-only attacks.

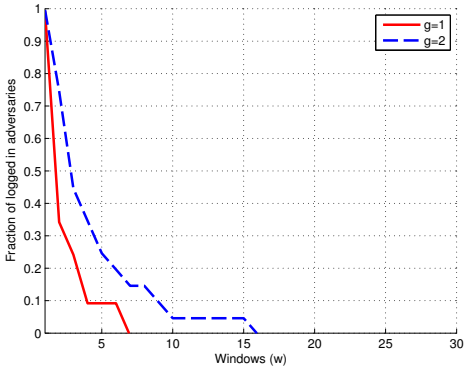
In all four cases, we use data from the 20 user sessions. As before, we use the leave-one-user-out approach: for a given session, we train Interaction Classifier using the bracelet traces from the 76 samples from the remaining 19 sessions. For each type of attack, we then apply the classifier for the corresponding trace in the current sample. Thus, the results for each attack scenario is the aggregated result of 20 classifications.

**Naïve all-activity**: Figure 6a presents the average False Positive Rate (FPR) for threshold values ($m$) between 50% and 70%, and for window sizes ($w$) in the 5-30 range. The FPR represents the fraction of authentication windows in which the attacker is mistaken for the victim, i.e., a large enough fraction of interactions are evaluated as matching. The FPR values range from 50-80% with a lenient threshold of 50%, and from 15-35% with a strict threshold of 70%. For example, with $m = 70\%$ and $w = 20$, less than one fifth of the attackers' authentication windows are correct.

We choose the same threshold and window size as previously described ($m = 60\%$ and $w = 20$), and determine the fraction of logged in users as a function of the number

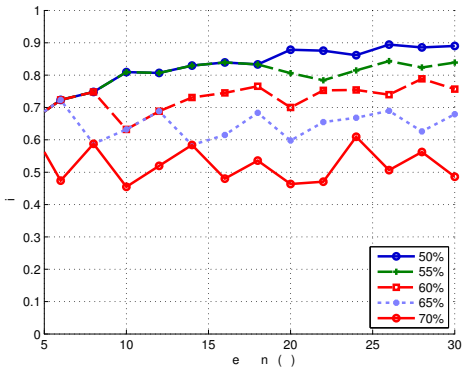



(a) Average FPR for different threshold ($m$) values. Fraction of attacker windows that are classified as matching with the bracelet.

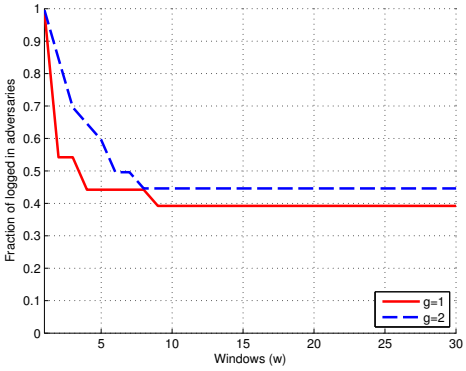



(b) Fraction of attackers remaining logged in after ($n$) authentication windows, for different grace periods ($g$).

Fig. 6: Results for **naïve all-activity** attackers. Naïve all-activity attackers try to replicate all mouse-hand movements.




(a) Average FPR for different threshold ($m$) values. Fraction of attacker windows that are classified as matching with the bracelet.




(b) Fraction of attackers remaining logged in after ($n$) authentication windows for different grace periods ($g$).

Fig. 7: Results for **opportunistic keyboard-only** attackers. Opportunistic keyboard-only attackers choose to replicate only a part of the keyboard movements of the victim.

of authentication windows. This represents how long the attackers successfully remain logged in. Figure 6b depicts this fraction for $g = 1, 2$ . The FPR of 43% from Figure 6a translates to all users eventually being logged out. With a strict grace period ($g = 1$) all attackers are logged out by the seventh authentication window, whereas with $g = 2$ one attacker remains logged in until window 16 (all others fail at window 10 at the latest). The victim in this one case had very slow interactions, which made them easier to mimic. The corresponding number of windows in the ZEBRA paper [23] were 2 and 4.

The naïve all-activity attacker is comparable to the attacker modeled in [23]. However, the performance of our system against such an attacker (as summarized in Figures 6a and 6b) is more lenient than the corresponding figures reported in [23]. Nevertheless, we can use the results for the naïve all-activity attacker as a baseline to compare against more sophisticated or smart attacker strategies we study next.

**Opportunistic keyboard-only**: We now consider an attacker who opportunistically mimics only a subset of the typing interactions. Figure 7a presents average FPR for different threshold values and window sizes. The FPRs are now noticeably higher. A threshold of $m = 60\%$ and a window size of $w = 20$ now

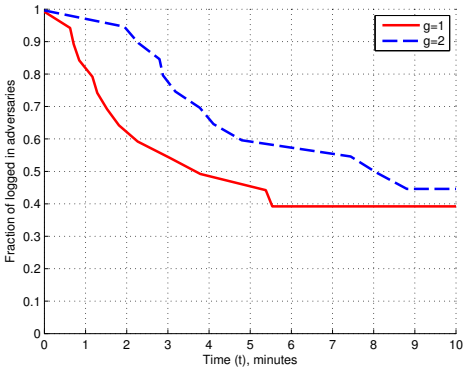



Fig. 8: **Opportunistic keyboard-only** attacker: Fraction of attackers remaining logged in after ($t$) minutes (with $w = 20, m = 60\%$), for different grace periods ($g$).

produces an FPR of 70%. Even with a stricter threshold of 70%, in around half of the windows, attacker interactions are incorrectly evaluated as matching the victim's interactions. In summary, windows are misclassified as correct ones roughly 20 percent points more often with an opportunistic keyboard-only attacker, compared to a naïve all-activity attacker.

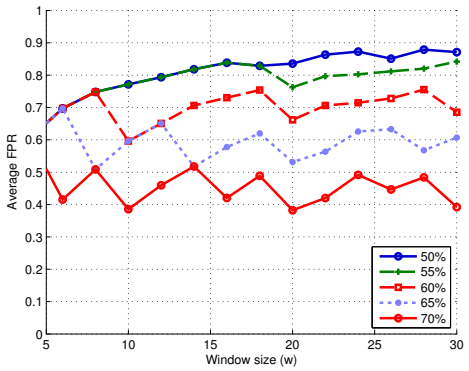
These high FPRs translate to almost half of the attackers

remaining successfully logged in for the whole duration of the experiment. Figure 7b depicts the fraction of logged in attackers as a function of the number of authentication windows, using $g = 1, 2$. Figure 8 shows the same information in terms of minutes. In terms of remaining successfully logged in, the advantage of an opportunistic keyboard-only attacker (Figure 7b) over the naïve all-activity attacker (Figure 6b) is statistically significant (Wilcoxon signed-rank test, $z = -2.928$ and $p = 0.003 \ll 0.05$) with medium effect size ($r = -0.46$). In other words, keyboard-only attackers remain logged in statistically longer than all-activity attackers. Using $g = 1$ results in 40% of the attackers remaining logged in throughout the experiment. A grace period of $g = 2$ increases this to 45%.

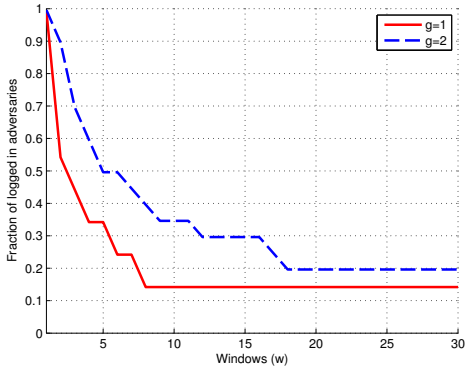
Given that an opportunistic keyboard-only attacker can do significantly better than the naïve all-activity attacker, we conclude that the attack scenario used in [23] to demonstrate the security of ZEBRA is *not the most favorable setting* for the adversary. Also, in our experiments the opportunistic attackers reproduced around 60% of the victims' typing interactions, reaching typing speeds of 20-40 words/minute. Even at this high typing rate, 40-45% of attackers were able to successfully evade detection throughout the experiment. A more conservative strategy would naturally increase the attacker success rates closer to 100%. To clarify, the number of interactions generated per unit of time is not bound to the typing speed: ZEBRA concatenates consecutive typing events into a single typing interaction of up to 1s in length. Victims who type slowly may give the attacker more time to mimic. For a malicious attacker, even a short period is enough to cause damage to the system. The attacker can, for example, mount a USB drive and execute a script from the drive in mere seconds.

**Other Attacks:** Having demonstrated that opportunistic keyboard-only attacks are effective, we now consider two variations. First we ask whether the opportunistic approach can be extended successfully to mimicking all types of activities rather than just typing. Figures 9a and 9b summarize the performance of the **opportunistic all-activity** attack. Compared to Figure 7a, average FPR values in Figure 9a are somewhat worse for the attacker. This results in opportunistic all-activity attackers being logged out at a higher rate compared to opportunistic keyboard-only attackers (although this is not statistically significant, with $z = -1.082, r = -0.17$ and $p = 0.279 > 0.05$). This is not surprising since mimicking all types of interactions is likely to be harder than mimicking typing interactions only. Nevertheless, opportunistic all-activity attackers are somewhat more successful than naïve all-activity attackers (but again not statistically significant, with $z = -1.514, r = 0.24, p = 0.130 > 0.05$). For example, with $g = 1$, all naïve all-activity attackers are logged out after 7 windows, while 25% of the opportunistic all-activity attackers succeed in remaining logged in.

Next we consider the question whether the inability of the attacker to see the victim hampers his ability to circumvent ZEBRA. Figures 10a and 10b summarize the performance




(a) Average FPR for different threshold ($m$) values. Fraction of attacker windows that are classified as matching with the bracelet.



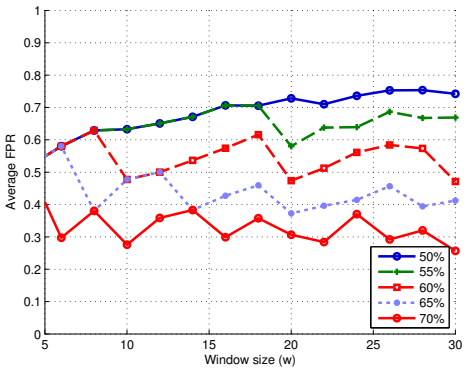

(b) Fraction of attackers remaining logged in after ($n$) authentication windows for different grace periods ($g$).

Fig. 9: Results for **opportunistic all-activity** attackers. Opportunistic all-activity attackers replicate easy mouse-hand movements of the victim.

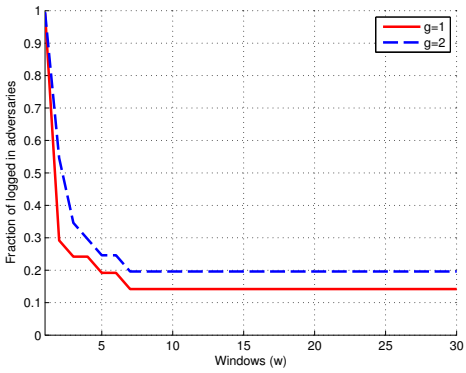
of an **audio-only opportunistic keyboard-only** attacker. This attack is in line with prior attacks based on keyboard acoustic emanations [3], [31]. Prior attacks aimed at recognizing the keystrokes based on their sounds, while our attack attempts to recognize typing/mouse activities based on their sounds. One key difference is that our attack is manual, whereas prior attacks were automated (in fact, it seems that prior attacks can not be performed manually since a human attacker may not be able to distinguish between sounds of different keys). As such, our attack may be viewed as a new form of acoustic emanations attack targeted at the ZEBRA system.

Again, we see that such an attack is less successful than an opportunistic keyboard-only attacker who is able to see his victim. However, it is still more successful than a naïve all-activity attack. Again, with $g = 1$, 15% of the audio-only opportunistic keyboard-only remain logged in after 6 windows.

Thus, we conclude that an attacker adopting an opportunistic approach can do better in circumventing ZEBRA than by naïvely mimicking all interactions. This holds even when the attacker is hampered by not having visual access to the victim. An opportunistic keyboard-only attacker performs significantly better than a naïve all-activity attacker.




(a) Average FPR for different threshold ($m$) values. Fraction of attacker windows that are classified as matching with the bracelet.




(b) percentage of attackers remaining logged in after ($n$) authentication windows for different grace periods ($g$).

Fig. 10: Results for **audio-only opportunistic keyboard-only** attackers. Audio-only opportunistic keyboard-only attackers eavesdrop on the victim and type only when they hear the victim typing.

## VI. Strengthening ZEBRA

Opportunistic attacks against ZEBRA succeed because of the fundamental flaw in its design: it allows the adversary to control both interaction sequences Authenticator receives as input. First, the adversary has full control over the actual interaction sequence as he can choose the type and order of his terminal interactions. Second, he can indirectly influence the predicted interaction sequence as his terminal inputs cause Interaction Extractor to choose the times at which the victim's bracelet data is segmented and fed to the Interaction Classifier to generate the predicted interaction sequence.

We can cast this as a general problem of *tainted input*: accepting data which can be incorrect or outright malicious, and performing security-critical actions based on it. This is a common issue in any application or on-line service accepting input from potential adversaries. There are typically three counter-measures: (1) augmenting with trusted input, (2) marking untrusted input as tainted and performing taint tracking, or (3) sanitizing untrusted input before using it. As the sole purpose of the interaction sequences is authentication, taint tracking is not applicable in our case. Thus, we consider the other two potential solutions: using trusted input and input sanitization.

**Augmenting with Trusted Input**: Instead of allowing the terminal input to fully determine when Authenticator compares the two interaction sequences, a *fundamental* fix is to base this determination additionally on bracelet data which is not under the control of the attacker. This would require inferring the predicted interaction sequence continuously from the bracelet data even when the terminal observes no actual interaction. If the predicted interaction sequence suggests that the user is interacting with a terminal, but no corresponding actual interaction is observed, Authenticator should output "Different User". This presupposes that the Interaction Classifier has very high precision (which we discuss below). Requesting data from the bracelet continually, rather than on demand, might lead to unwanted deauthentication if the event is not recognized.

Augmenting ZEBRA with Bluetooth proximity measurements means that we have another way of assurring ourselves that the user is nearby. We noticed that typical bluetooth signal strengths are within -5dB for users immediately close by, e.g. working at the terminal. Similarly, users walking nearby the terminal tend to have signals strengths within -15dB. Based on this, a three-level proximity calculation could be developed, classifying the proximity of the user as *immediate*, *near* or *far* based on the Bluetooth signal strength. Users that are perceived as being near or far could have progressively increased authentication thresholds, e.g. increasing the threshold from 70% to 80% in case of near distance and further to 90% in case of far distance. This would make mimicking attacks more difficult, because the attacker needs to be very close to the victim in order to have a lenient threshold.

In a centralized (multi-terminal) environment, it may be possible to use successful login events as an input for triggering deauthentication: a central system could recognize when a user logs into a terminal and automatically deauthenticate him) from any other terminal where he has an active logged-in session.

**Sanitizing Untrusted Input**: Input sanitization can take the form of whitelisting (accepting specific well-formed inputs only) or blacklisting (rejecting a set of known malicious input patterns). Authenticator has two inputs that need to be sanitized: the actual interaction sequence and the predicted interaction sequence.

For example, one could attempt to prevent our opportunistic keyboard-only attacker by adopting a whitelisting approach of only accepting actual interaction sequences which contain *multiple types* of interactions, such as requiring periodic MKKM interactions interspersed with typing. However, since many legitimate user sessions can involve typing-only sequences, this remedy will violate the zero-effort requirement.

ZEBRA could also use blacklisting where certain types of input data can immediately trigger deauthentication. For example, if an input stream can reliably indicate the user standing up and walking away from the terminal, it can

trigger deauthentication. Augmenting the bracelet data we currently use (accelerometer and gyroscope) with additional information, like heart-rate data available on many current smartwatches, can be used for this purpose. However, these fixes can seem privacy-invasive for some users.

**Further Instances of Tainted Input**: We identify additional types of input interactions that an adversary can use to defeat ZEBRA. As the bracelet is assumed to be worn on the mouse-controlling (e.g., right) hand, ZEBRA records an MKKM interaction after mouse activity only if it observes a keypress event on that side of the keyboard. This is done to reduce false negatives arising from a user who types with the keyboard-only hand without removing his mouse-controlling hand from the mouse. Again, such a design decision introduces a vulnerability: for example, in the case of a right handed victim, the attacker can type using only the left and middle parts of the keyboard (approximately 60% of the keys) while the victim continues to use the mouse. Having previously recorded an interaction involving the mouse, ZEBRA will leave out all such subsequent typing from the segments it considers for comparison. This could be mitigated by blacklisting long typing sequences involving keys in the middle and non-dominant parts of the keyboard as such sequences are not typical in normal workstation usage.

Another such vulnerability is when $\mathcal{A}$ interacts by only moving and clicking the mouse. No event gets reported for these activities and consequently an adversary can potentially do much harm, for example by copy-pasting words appearing in the screen. Mare et al. [23] report that they did not consider mouse movement and click events as interactions because "they did not contribute to ZEBRA's performance." However, including them would seem the most feasible defense. As we can see in our examples, pre-mature optimization motivated by privacy (such as not collecting data under certain scenarios) may introduce security vulnerabilities.

**Making the System Work in Real-time**: It is well-known that on-line systems always bring new information of the usability, compared to off-line analysis. When we experimented with our end-to-end implementation in real-time, we noticed that the original 3-class classifier (typing, scrolling and MKKM) systematically identified bracelet measurements during walking and standing ("upright") as typing interactions. Similarly, measurements while the bracelet was simply lying on the table ("idle") were classified as scrolling interactions. The similarity between the hand movements in these pairs of events leads to similar magnitude-based features. Based on these observations, we extended the original classifier to account for the new types of "interactions": idle and upright. The new classes were added as a post-processing step: a one hundred random tree ensemble was learned first (random forest), each tree contributing by voting between one of the three original classes. These votes were fed to a C4.5 decision tree, which learned decision thresholds for the five classes. We noticed that the performance of the resulting *real-time system* improved a great deal. We observed an overall improvement in both the accuracy and the ability to generalize to new users.

**Improving Machine Learning**: Mare et al. [23] make use of accelerometers and gyroscopes that report measurements in three dimensions but use only magnitude values calculated from a single dimension. ZEBRA, and other similar techniques in general, can be extended to use measurements from all three dimensions. The gravity component in individual axes can be eliminated with a low-pass filter [2]. The added information from statistical measures in any individual direction can help in the discrimination of the classes, increasing the accuracy of the classifier in normal usage. With a better classifier we can raise the threshold ($m$) of authenticating a user interaction, lowering the FPR, while increasing the FNR. An acceptable FNR level can then be found as a compromise with receiver operating characteristic (ROC) curves, which shows the trade-off between TPR and FPR.

The Scrolling events were more difficult to identify compared to others in our experiments (Table I). So improving the accuracy of these predictions is of interest. Further feature engineering can increase the classification ability. Feature selection algorithms can select robust features that generalize the decision rules well. Feature selection can also help in increasing the battery life of the bracelet, since less information needs to be transmitted over Bluetooth from the bracelet to the computer for classification.

## VII. Discussion

Despite our attempts to reproduce the implementation described in [23], differences remain. Although our implementation achieves lower FNR for legitimate users, it incurs somewhat longer delays in logging out naïve attackers. We were unable to reproduce the high rate of user interactions reported in [23]. Despite these differences, the main result of our work holds because it is *comparative*: we demonstrated that in our system, attackers adopting opportunistic strategies can significantly outperform a naïve all-activity attacker. Such a comparative result will hold in any implementation of ZEBRA, including [23], despite any differences between implementations.

**Impact of Data Set**: One contributor in performance differences could be a methodological difference we discovered with the original paper. The authors note that one user's "wrist movement during keyboard and mouse interaction were very different compared to the other subjects", and one of the test users is logged out almost immediately. It is likely that a large fraction of this user's authentication windows are thus incorrectly classified, amounting to 1/20, i.e. 5 percent points difference in FPR between their experiments and ours.

One potential explanation is that in [23] only one of the users was left-handed, which may result in differences for this one user. The leave-one-user-out classifier training may also exaggerate this as it results in the classifier being trained with data from only right-handed users, but tested with data from the left-handed user. However, without access to the original test data, this cannot be verified.

**Impact of Sampling Rate**: A notable difference lies in the sampling rate of the bracelet. We chose to use commercial off-the-shelf smartwatches as bracelets because they are general-purpose devices readily available for a much larger audience and thus a realistic choice for deployment. In such devices, the underlying sensor hardware limits the maximum sampling rate, typically 100-200 Hz on newer devices. Our LG smartwatch supported a sampling rate of 200 Hz. This is less than the 500Hz special-purpose Shimmer bracelet used in [23].

The choice of sampling rate has an impact on power consumption [6]. On Android, the sampling rate can be set to lower levels to save energy at the cost of reduced accuracy. The features we collect are mostly statistical measures calculated from the distribution of magnitude values measured during the event and should be quite stable as long as there are enough data points to calculate the values from.

To evaluate the effect of sampling rates, we collected a small dataset from normal computer usage with 200 Hz sampling rate. We downsampled it to 100 Hz, 50 Hz and 25 Hz data sets by passing every second, fourth or eight measurement signal to Segmenter. The datasets were generated using the same data: the number of features and the number of events are the same at all frequencies, but the number of measurement signals used to calculate the features were different. We noticed that some features (e.g. skewness) could frequently not be calculated for short events at low frequencies because Segmenter could not pass enough measurements signal values to Feature Extractor. Sample skewness needs at least three values to be calculated. As a rule of thumb, the frequency of the bracelet needs to be at least $f_{min} = s_{min}/d_{min}$ to catch enough measurements for feature calculation, when $s_{min}$ (3) is the minimum amount of signal measurements needed to calculate all features and $d_{min}$ (25 ms) is the minimum duration of a classifiable event. With our end-to-end system parameter settings this would be $f_{min} = 120$ Hz. For devices operating at lower sampling rates, the minimum acceptable duration of interactions should be increased accordingly.

Typically lower sampling rates increase the noise in features, which in turn changes class boundaries. We expect minor classes to get misclassified as major classes more frequently. In the worst scenario, everything gets classified as the major class (typically this would be typing in our scenarios). Lower sampling rates would increase the FPR in this way. This is not the case in 200 Hz, as can be seen in Table I (Section IV-C). We experimented on our real-time system with one such lower rate (20 Hz), at which the traces contain very little information for each interaction causing ZEBRA to become insensitive to synchronization delays as long as 1s. At higher rates above 120 Hz there was no noticeable difference. Therefore, we conclude that while ZEBRA is unreliable at very low sampling rates, its performance was found to be steady at or above 120 Hz.

## VIII. Related Work

User authentication is commonly based on three different factors: something you know, something you have, and something you are. Many traditional authentication methods rely on the first two. Passwords still remain very popular, and they are often complemented with some sort of physical tokens, such as RSA SecurID [14]. The downside is the need to deploy and carry these tokens.

A large body of research has considered biometric authentication. Examples include the use of fingerprint [11], hand [21], iris [10], facial [4] or blood vessel information [30]. Biometric authentication is attractive because they reduce the user burden by removing the need to memorize secrets or having to carry external tokens. However, these schemes can still be vulnerable to spoofing, and introduce new issues such as the problem of revocation and raise privacy concerns. Also, traditional biometric authentication is not transparent to the user.

The desire to minimize the user burden of authentication has led to a quest for transparent and continuous authentication schemes that can be "zero-effort." One approach uses proximity-based authentication where the presence of a personal device is used to authenticate the user. Such schemes can be based on RFID, NFC, Bluetooth or even WiFi signal strength. The appeal is the possibility to use existing devices seamlessly, but unfortunately the drawbacks include limited accuracy and vulnerability to spoofing and replay attacks [18] [19] [15].

Behavioral biometrics consider behavior intrinsic to specific individuals. A common example is gait, identified from video or acceleration information. Gafurov et al. [16] perform authentication based on a user's gait, which is characterized by recorded accelerations from a hip-worn device. The same author also considers [17] spoofing attacks against gait-based authentication. A subset of these behavioral biometrics are keystroke and typing based authentication schemes. In an early work, Joyce et al. [20] and Monrose et al. [25] [24] identify users based on their typing rhythm. They consider the inter-key latencies and are able to effectively authenticate users. More recently, Ahmed et al. consider mouse dynamics for authentication [1]. [13] distinguishes users based on how they input touch patterns into a smartphone. However, Tey et al. [27] show how through training, attackers can learn to defeat keystroke biometrics based authentication.

Combining multiple types of transparent authentication schemes, such as the proposal by Riva et al. [26], can improve the overall performance. But the design of such systems is complex and remains an open research problem.

## IX. Conclusions

ZEBRA is an interesting and useful approach as a zero-effort deauthentication system. We identified a subtle design flaw in this approach, which is (1) easier for the human operators to perform and (2) more robust, compared to the naïve attacks studied by the authors of the ZEBRA scheme [23]. We demonstrated that a malicious adversary who adopts an opportunistic strategy can defeat ZEBRA. This is at odds with the positive results reported in [23] but is explained by their attackers using a naïve strategy of trying to mimic

all interactions of a victim. Our attack is done in a typical usage scenario. While physical mitigations, such as visual barriers, might make our specific attack less successful, the underlying vulnerability still stands. Although susceptible to opportunistic adversaries, ZEBRA still performs well against accidental misuse by innocent adversaries, which is possibly the most likely threat in scenarios that ZEBRA was originally designed for. However, systems are often used in contexts that the designers did not originally envisage. Therefore, we believe that recognizing the limits of the original design of ZEBRA against malicious adversaries is the first step towards strengthening its resistance so that it can be used in scenarios where malicious adversaries pose a significant threat. The approaches we identified in Section VI can help secure ZEBRA without losing its desirable properties. We are developing these approaches further in our current work. More generally, we showed that subtle design assumptions based on premature usability and privacy considerations can adversely impact security of a system. We also highlight the importance of ensuring that adversary models used in analyzing the security of systems are realistic and do not underestimate attacker capabilities.

**Acknowledgments**: This work was supported in part by the Academy of Finland "Contextual Security" project (274951), NSF grant CNS-1201927 and a Google Faculty Research Award. We thank Shrirang Mare for explaining ZEBRA design parameters, and Hien Truong and Babins Shrestha for discussions on transparent deauthentication.

## REFERENCES

[1] Ahmed Awad E Ahmed and Issa Traore. A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4(3):165–179, 2007.

[2] android.com. Android API: SensorEvent. http://developer.android.com/reference/android/hardware/SensorEvent.html.

[3] Dmitri Asonov and Rakesh Agrawal. Keyboard Acoustic Emanations. In *Proc. IEEE Symposium on Security and Privacy*, 2004.

[4] Charles Beumier and Marc Acheroy. Automatic 3D face authentication. *Image and Vision Computing*, 18(4):315–321, 2000.

[5] BlueProximity. SourceForge Project. http://sourceforge.net/projects/blueproximity/.

[6] Agata Brajdic and Robert Harle. Walk detection and step counting on unconstrained smartphones. In Friedemann Mattern, Silvia Santini, John F. Canny, Marc Langheinrich, and Jun Rekimoto, editors, *The 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp '13, Zurich, Switzerland, September 8-12, 2013*, pages 225–234. ACM, 2013.

[7] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[8] Bump Technologies. Bump application. Android and iTunes App stores. http://bu.mp.

[9] Ming Ki Chong, Rene Mayrhofer, and Hans Gellersen. A survey of user interaction for spontaneous device association. *ACM Comput. Surv.*, 47(1):8, 2014.

[10] Siew Chin Chong, Andrew Beng Jin Teoh, and David Chek Ling Ngo. Iris authentication using privatized advanced correlation filter. In *Advances in Biometrics*, pages 382–388. Springer, 2005.

[11] T Charles Clancy, Negar Kiyavash, and Dennis J Lin. Secure smartcard-based fingerprint authentication. In *Proceedings of the 2003 ACM SIGMM workshop on Biometrics methods and applications*, pages 45–52. ACM, 2003.

[12] Mark D. Corner and Brian D. Noble. Zero-interaction authentication. In *Proc. 8th annual international conference on Mobile computing and networking*, MobiCom '02, pages 1–11, New York, NY, USA, 2002. ACM.

[13] Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. Touch me once and I know it's you!: implicit authentication based on touch screen patterns. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 987–996. ACM, 2012.

[14] EMC Corporation. RSA SecurID. http://www.emc.com/security/rsa-securid/index.htm.

[15] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical NFC peer-to-peer relay attack using mobile phones. In *Radio Frequency Identification: Security and Privacy Issues*, pages 35–49. Springer, 2010.

[16] Davrondzhon Gafurov, Kirsi Helkala, and Torkjel Søndrol. Biometric gait authentication using accelerometer sensor. *Journal of computers*, 1(7):51–59, 2006.

[17] Davrondzhon Gafurov, Einar Snekkenes, and Patrick Bours. Spoof attacks on gait authentication system. *Information Forensics and Security, IEEE Transactions on*, 2(3):491–502, 2007.

[18] Gerhard P Hancke. A practical relay attack on ISO 14443 proximity cards. *Technical report, University of Cambridge Computer Laboratory*, pages 1–13, 2005.

[19] Gerhard P Hancke. Practical attacks on proximity identification systems. In *Security and Privacy, 2006 IEEE Symposium on*, pages 6–pp. IEEE, 2006.

[20] Rick Joyce and Gopal Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2):168–176, 1990.

[21] Ajay Kumar and K Venkata Prathyusha. Personal authentication using hand vein triangulation and knuckle shape. *Image Processing, IEEE Transactions on*, 18(9):2127–2136, 2009.

[22] Shrirang Mare. personal communication.

[23] Shrirang Mare, Andres Molina-Markham, Cory Cornelius, Ronald A. Peterson, and David Kotz. ZEBRA: zero-effort bilateral recurring authentication. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 705–720. IEEE Computer Society, 2014.

[24] Fabian Monrose, Michael K Reiter, and Susanne Wetzel. Password hardening based on keystroke dynamics. *International Journal of Information Security*, 1(2):69–83, 2002.

[25] Fabian Monrose and Aviel D Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation computer systems*, 16(4):351–359, 2000.

[26] Oriana Riva, Chuan Qin, Karin Strauss, and Dimitrios Lymberopoulos. Progressive authentication: Deciding when to authenticate on mobile phones. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 301–316. USENIX Association, 2012.

[27] Chee Meng Tey, Payas Gupta, and Debin Gao. I can be you: Questioning the use of keystroke dynamics as biometrics. The 20th Annual Network & Distributed System Security Symposium (NDSS 2013), 2013.

[28] Hien Thi Thu Truong, Xiang Gao, Babins Shrestha, Nitesh Saxena, N.Asokan, and Petteri Nurmi. Comparing and fusing different sensor modalities for relay attack resistance in zero-interaction authentication. In *IEEE International Conference on Pervasive Computing and Communications, PerCom 2014, Budapest, Hungary, March 24-28, 2014*, pages 163–171, 2014.

[29] Shaun W. Van Dyken et al. Multi device pairing and sharing via gestures. United States Patent Application 20140149859, May 2014.

[30] Masaki Watanabe, Toshio Endoh, Morito Shiohara, and Shigeru Sasaki. Palm vein authentication technology and its applications. In *Proceedings of the biometric consortium conference*, pages 19–21, 2005.

[31] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Trans. Inf. Syst. Secur.*, 13(1):3:1–3:26, November 2009.

TABLE III: Features used in this paper.

| **Feature** | **Description** |
|---|---|
| Mean | mean value of signal |
| Median | median value of signal |
| Variance | variance of signal |
| Standard Deviation | standard deviation of signal |
| MAD | median absolute deviation |
| IQR | inter-quartile range |
| Power | power of signal |
| Energy | energy of signal |
| Peak-to-peak | peak-to-peak amplitude |
| Autocorrelation | similarity of signal |
| Kurtosis | peakedness of signal |
| Skewness | asymmetry of signal |

## APPENDIX

The parameters we use in our end-to-end system are listed in Table II.

TABLE II: Parameters and their values used in this paper.

| **Parameter** | **Value** | **Rationale** |
|---|---|---|
| Min. duration[a] | 25 ms | [22] |
| Max. duration[b] | 1 s | [23] |
| Idle threshold[b] | 1 s | [22] |
| Window size ($w$) | 5-30 | [23] |
| Match threshold ($m$) | 50-70% | [23] |
| Overlap fraction ($f$) | 0 | Estimated[c] |
| Grace period ($g$) | 1-2 | [23] |

[a]For scrolling, also a minimum of 5 recorded events.

[b]For MKKM, a max. duration and idle threshold of 5s.[22]

[c]Estimate based on reported [23] times & authentication windows needed for logging out users.

We consider the same features as in [23], listed in Table III. These are extracted from segments of sensor readings and used to classify interactions.