ZEBRA: Zero-Effort Bilateral Recurring Authentication

Shrirang Mare*, Andres Molina-Markham*, Cory Cornelius†, Ronald Peterson* and David Kotz*
*Institute for Security, Technology, and Society, Dartmouth College
†Intel Labs

***Abstract*—Common authentication methods based on passwords, tokens, or fingerprints perform one-time authentication and rely on users to log out from the computer terminal when they leave. Users often do not log out, however, which is a security risk. The most common solution, inactivity timeouts, inevitably fail security (too long a timeout) or usability (too short a timeout) goals. One solution is to authenticate users continuously while they are using the terminal and automatically log them out when they leave. Several solutions are based on user proximity, but these are not sufficient: they only confirm whether the user is nearby but not whether the user is actually using the terminal. Proposed solutions based on behavioral biometric authentication (e.g., keystroke dynamics) may not be reliable, as a recent study suggests.**

To address this problem we propose Zero-Effort Bilateral Recurring Authentication (ZEBRA). In ZEBRA, a user wears a bracelet (with a built-in accelerometer, gyroscope, and radio) on her dominant wrist. When the user interacts with a computer terminal, the bracelet records the wrist movement, processes it, and sends it to the terminal. The terminal compares the wrist movement with the inputs it receives from the user (via keyboard and mouse), and confirms the continued presence of the user only if they correlate. Because the bracelet is on the same hand that provides inputs to the terminal, the accelerometer and gyroscope data and input events received by the terminal should correlate because their source is the same – the user's hand movement. In our experiments ZEBRA performed continuous authentication with $85\,\%$ accuracy in verifying the correct user and identified all adversaries within $11\,\mathrm{s}$. For a different threshold that trades security for usability, ZEBRA correctly verified $90\,\%$ of users and identified all adversaries within $50\,\mathrm{s}$.

I. Introduction

Desktop computers (also called *computer terminals* or simply *terminals*) are still being used in large numbers at workplaces and at homes, often by multiple users. To prevent unauthorized access users *authenticate* themselves before using the terminal (e.g., by logging in with username and password) and *deauthenticate* (i.e., log out) after their use. This important deauthentication step, however, is overlooked by most authentication schemes. Common schemes such as password-based or fingerprint-based authentication provide one-time authentication and rely on the users to log out. Unfortunately, users often do not log out, they either forget to log out or intentionally do not log out to avoid logging in again. Although deauthentication is important for many different devices, our focus in this work is to address the deauthentication problem on computer terminals; future extensions will support laptops, tablets, and phones.

The consequence of not logging out from a terminal can be severe: an adversary with access to your terminal can snoop through your private information, modify or delete your data, or steal your credentials to take actions on your behalf. Even in a non-adversarial setting, other authorized users could accidentally misuse your account if you forget to log out. For instance, Koppel et al. [1] report that physicians frequently enter data into the wrong patient's record because they thought the open record belonged to the patient they were treating; in fact, while they were away from the terminal another physician used that terminal to update a different patient's record and forgot to log out. Sometimes clinicians leave terminals intentionally logged in, as a professional courtesy, so that the next user does not have to log in. This deauthentication problem is a major concern in a busy multi-user environment where users have to authenticate and deauthenticate often, e.g., when users' use of terminals is interlaced or when they have to use a terminal multiple times but for short durations. For example, in a clinical inpatient setting where physicians, nurses, residents, and medical students come and go, check on patients, and use any available terminal to view and update patient medical records; or in a busy restaurant or retail shop where employees share kiosks to place orders and manage bills.

Even in workspaces where users have their own personal terminals, deauthentication is an important problem. Users either forget to log out or intentionally do not log out as a workaround to avoid logging in later; they find password-based authentication (the most common authentication method) tedious and time consuming. Thus, users often log in once when they arrive at work and log out when they leave, but they do not log out during their shift when they step away from the terminal, leaving their terminal vulnerable to snooping and attacks by a co-worker or a passerby.

The most common solution to the deauthentication problem is 'timeouts', i.e., to execute automatic logout after inactivity for a *timeout period*. The problem with this approach is that a single timeout period does not work for everyone and often timeout periods are blind to context [2], [3]. Another approach is to use a proximity sensor that detects a users' departure and log them out, but these sensors are unreliable in crowded environments [4].

One way to automate the deauthentication process is to continuously authenticate the current user and, when the user changes, deauthenticate the previous user and ask the new

user to authenticate herself. There are two goals that make this method challenging. First, continuous authentication should be passive (i.e., without requiring explicit action by the user) and unobtrusive so that it is not a burden on the user. Second, the system should quickly identify that the user has departed or changed without relying on users to log themselves out, because users often forget to log out. Computer use that is frequent, irregular, and short-lived – as is the case in hospitals – makes continuous authentication particularly challenging.

Our approach is to continuously authenticate a user based on her interactions with a terminal by monitoring her hand movements and comparing them with her inputs to the terminal using input devices (i.e, the keyboard and mouse). As with behavioral biometrics based on keystroke and mouse dynamics, our approach is based on the user's interactions – but there is an important distinction. Behavioral biometrics rely on **how** the user does a particular interaction (e.g., how the user types or how the user moves a mouse) and hence require user-specific training and typically require long periods of observation to authenticate the user. Our approach relies on **what** interactions the user does when using a terminal and hence does not require user-specific training or long periods of observation to authenticate the user. We confirm the user's continued presence by observing what the user is doing from two different sources and comparing those observations; we call this *bilateral* authentication. This approach complements any method that may be used for initial authentication, such as a password, a token, or a fingerprint biometric.

Zero-Effort Bilateral Recurring Authentication, or ZEBRA, monitors a user's hand movements via a bracelet worn on their wrist that they use to control the mouse. This bracelet is registered to the user, like any authentication token, so its presence should imply the presence of the associated user. ZEBRA goes beyond mere proximity, however. After logging in (using additional credentials) the user interacts with the terminal and the bracelet records the user's hand movements using built-in accelerometer and gyroscope sensors and transmits their data to the terminal over a short-range radio (e.g., Bluetooth). The terminal then compares the user's hand movements with the inputs it observes and confirms the presence of the user if they correlate. For example, when the user clicks the mouse and then starts typing (with both hands) his hand used to control the mouse (bracelet hand) moves from the mouse to the keyboard; when the user scrolls using the mouse scroll-wheel his hand is relatively stationary. It is these kinds of hand motions that the terminal expects for inputs that it receives from the user. The core idea of ZEBRA is that if the bracelet is on the same hand that provides inputs to the terminal then the accelerometer and gyroscope data (from the bracelet) and the terminal input events should correlate because their source is the same – the user's mouse hand movement. Conversely, if these movements no longer correlate, the terminal infers that a different person is now using the terminal and can take action (e.g., to lock the screen, log out the former user, or require initial authentication for the new user).

We make three main contributions. First, we introduce a new type of authentication, *bilateral* authentication, which falls into a new category of authentication of '**what** the user does when interacting' with a terminal. It is worth mentioning again that this is distinct from behavioral biometrics that fall into the category '**how** the user interacts'. Second, we propose ZEBRA, a novel mechanism to continuously authenticate the current user passively and unobtrusively, and to automatically deauthenticate the user. We further describe how ZEBRA can be used to improve the initial authentication process (e.g., username and password). Third, we evaluate ZEBRA's performance with a user study and demonstrate strong results.

II. Background and related work

Passwords are one of the oldest and the most common authentication schemes. Passwords are convenient because users do not have to carry anything, they are intuitive, and are efficient to use. However, the plight of passwords is well documented [5], [6]. Users find it hard to remember strong passwords, and they use workarounds such as choosing weak passwords, sharing them, reusing them, writing them down, or *intentionally leaving their terminals unlocked* so that they do not have to enter the password again [4], [7]–[12]. One problem with password-based authentication schemes is that there is no reliable and convenient way to deauthenticate users, so if a user leaves a terminal unlocked any passerby can access it. Organizations have tried timeouts for auto deauthentication, but efforts have failed because they are not reliable [2].

To address the limitation of a timeout approach to deauthenticate, some have proposed using proximity sensors that detect close proximity of a human, and when the user walks away, they detect the user's departure and log out the user if necessary. Proximity-based deauthentication, like timeouts, is blind to context, and one proximity setting may not work for all users. Moreover, as Sinclair et al. [4] report, these sensors were not reliable when they were tested in a hospital – they will trigger when someone walks past in the hallway and would sometimes log a user out when she was still using the terminal. Their deauthentication solution frustrated the users, who developed a work-around by covering the proximity sensor with an empty cup.

A proximity-based authentication scheme using a wearable token (e.g., ZIA [13]) also provides passive continuous authentication, but such schemes are not reliable in dense workspaces such as hospitals, where multiple authorized users may be near the terminal at the same time. Although proximity-based authentication schemes are well suited for single-user machine scenarios, their attack window lasts until the user walks out of the proximity range, which may be many

minutes. When there is a group of clinicians near a device, whom should the device authenticate? Sinclair et al. [4] note these shortcomings of proximity-based authentication mechanisms from their observations of clinicians' terminal use and their discussions with the clinicians.

Biometrics are convenient because users do not have to remember their credentials – they are always with them. However, biometrics can be stolen by physical observation or internal observation (from within the device) and they are hard to recover from theft or loss, because it may not be possible to change the biometric. Some biometrics such as fingerprint or iris require user input, which makes them unsuitable for continuous user authentication. Biometrics based on voice or face may be suitable for continuous authentication provided they can be captured easily and correctly without interrupting the user, which is not always possible when the user is not speaking or in front of her camera. Behavioral biometrics based on keystroke dynamics or mouse dynamics provide unobtrusive passive continuous authentication. However, they require a user-enrollment step and as a user's behavior changes they need to be re-enrolled, which may increase the maintenance cost of this scheme. Moreover, keystroke-based biometric is not resilient to theft or internal observation, as Meng et al. [14] show that an attacker with some training can successfully mimic a user's typing behavior. Rasmussen et al. [15] propose a pulse-based biometric for continuous authentication where a metal keyboard sends small electric current through the user's body and verifies the user based on the user's body's resistance to the current. Such an approach has significant deployment cost, because it requires modifying every input device. This scheme requires users' hands to be in-touch with the keyboard at the same time for the pulse to pass through the body, and this restricts how the user may use a keyboard.

Table I shows a comparative evaluation of ZEBRA with the authentication schemes described above, using the usability-deployability-security (UDS) framework [16]. UDS framework is actually for evaluating web authentication schemes, but some of its evaluation properties are applicable to authentication schemes for devices; we use some of those properties and 3 additional properties to compare continuous authentication schemes: *Verifying-Actual-User*, the scheme should verify who is actually using a terminal; *No-Constraint-on-Using-the-Device*, the scheme should work irrespective of how the user uses the device; and *Automatic-Deauthentication*, the scheme should automatically deauthenticate users. Due to lack of space we do not provide a detailed comparative evaluation here; we refer readers to our technical report for details [17].

The closest work is perhaps 'shake well before use' [18], in which a user shakes two devices to generate a shared encryption key between them. Also similar is the product Bump [19], wherein users bump their phones to exchange contacts. In these works, the same action (shaking or bumping) is observed by two different devices and compared to generate an encryption key or match two devices. In both these cases, the user must take explicit action, so these methods are not passive or unobtrusive. Furthermore, the signals being observed and compared are of the same type – accelerometer signals. In our work, we have two different types of signals to correlate – one is from the sensors in the bracelet and the other is a set of input events on a terminal.

Table I: Comparative evaluation of ZEBRA against other continuous authentication schemes.

Scheme	*Nothing-to-Carry*[1]	*Negligible-Cost-per-User*[1]	*Resilient-to-Physical-Observation*[1]	*Resilient-to-Internal-Observation*[1]	*Verifying-Actual-User*[2]	*No-Constraint-on-Using-the-Device*[2]	*Automatic-Deauthentication*[2]
ZEBRA		○	●	●	●	○	●
Passwords	●	●				●	
Proximity-based		○	●	●	○	●	●
Fingerprint	●		●			●	
Voice-based	●		●		○		●
Facial recognition	●				○	○	●
Keystrokes-based	●	○	●		●	○	●
Pulse-based	●		●	●	●	○	●

●= offers the benefit; ○= almost offers the benefit; *no circle*= does not offer the benefit. ||||= better than ZEBRA; ≡= worse than ZEBRA; *no pattern*= equivalent to ZEBRA.

[1]Properties from the UDS framework. [2]Additional properties, not in UDS.

Consider the following two use cases that motivate the need for ZEBRA.

Sally is a member of clinical staff in a hospital. She walks to a computer terminal and logs in to update her patient's record. She needs some more information from her colleague to update the record. She steps away to talk to that colleague, leaving the terminal open because she is planning to come back and update that record. Sally does not return before the timeout period expires, so the terminal automatically logs Sally out. In the meantime another clinician, Tina, logs into the same terminal and updates a record and leaves, again forgetting to log out. Soon Sally returns and finds that terminal open. She assumes that it is still her account and her patient's record, since she was using it earlier. Sally does not check whether it is indeed her account and her patient, nor does the system, and she accidentally updates the wrong patient's record under Tina's name. Incidents like these are not uncommon in hospitals [1], [3].

Claire, a chemical engineer, is authorized to alter the operation of a drug manufacturing plant using a terminal linked to the plants SCADA system. After logging in she gets an emergency call from her sister and walks around a

nearby corner to hear her sister better, knowing the terminal automatically logs out in 15 minutes. Jake, a biomedical engineer competing with Claire for a promotion, happens by and notices the open terminal is logged into Claire's account. He makes a subtle change in the plants operation that reduces efficiency and gets logged as Claire's doing, and then quickly walks away. Claire, the better and more honest engineer is passed over for promotion in favor of devious Jake because of her "mistake". The US Code of Federal Regulations, Title 21, Part 11 requires many FDA-regulated industries such as drug makers, medical device manufacturers, biotechnology companies, biologics developers and others to implement measures to control, monitor and report access to critical terminal control systems [20], [21]. Terminal timeouts are an important part of these protections but may be inadequate to prevent tampering.

III. System model

ZEBRA is designed to prevent intentional and accidental misuse of a user's account on a terminal. ZEBRA is not a method for initial authentication; rather, it compliments any existing initial authentication schemes by providing continuous authentication and automatic deauthentication. When a user logs in (e.g., by providing username and password), ZEBRA continuously authenticates the current user (i.e., verifies whether the current user is the same user who logged in), and when a different user starts using the same terminal while the current user is logged in, ZEBRA deauthenticates the current user, thereby preventing account misuse. In this section, we state the assumptions that we make for ZEBRA, its desired properties, and its adversary model.

Assumptions. We make the following assumptions.

1) We assume that each user wears a bracelet on the hand she uses to control the mouse interface. The bracelet has built-in accelerometer and gyroscope sensors and a wireless radio (e.g., Bluetooth) that it uses to communicate with the terminal. Today, many wrist-worn fitness devices meet these assumptions, demonstrating that such a device is feasible and can have long battery life.
2) Each bracelet is associated with a single user, and users do not share bracelets. This association can be implemented using a variety of approaches. For instance, one can use a biometric bracelet [22], or a user may be required to enter a PIN when she puts the bracelet on to activate it, and the bracelet would deactivate when it is removed from the wrist or after a period of time (e.g., 24 hours). This assumption is similar to the photo ID-cards used by many organizations. In other instances there might be a biometrically-authenticated station where employees check out bracelets at the start of a shift.
3) The bracelet and the terminal are already paired; they share encryption keys that they can use to secure their communication. Pairing is a one-time task and any suitable pairing method may be used [23]. In an enterprise setting, we assume administrative tools pair all bracelets with all terminals in a distributed fashion.
4) We assume that all communication between the bracelet and terminal is secured by other means (e.g., Bluetooth Low Energy or ANT+ protocols). The terminal does not communicate wirelessly to untrusted/unknown bracelets.
5) There exists an initial authentication scheme (e.g., username-password) that users use to log in to terminals. Once they log in, ZEBRA continuously verifies that the current user is the same user who logged in.

Desired properties. We desire ZEBRA to be

1) *Continuous:* It should continuously authenticate the current user as long as the user is logged in.
2) *Passive:* It should not require any explicit user intervention and should not interrupt the user.
3) *Unobtrusive:* It should be completely unobtrusive and should not invade the user's privacy; the user should be comfortable using the system.
4) *Quick:* It should be quick to identify when a user other than the logged-in user starts using the terminal so that it can deauthenticate the logged-in user to prevent any access misuse.
5) *Accurate:* It should not incorrectly deauthenticate a user nor falsely authenticate a user.
6) *User-agnostic:* It should not require any user-specific training.

Adversary model. We are primarily concerned with the threat of unauthorized access when the user forgets to log out when stepping away from the terminal, even if the user remains in the terminal's proximity doing other tasks (e.g., walking, writing, talking to someone, or working on another nearby terminal). If the user steps out of the radio's proximity range of the terminal one can use proximity-based solutions. We consider two types of adversaries. First, an innocent authorized user who wants to use a terminal for her own task: she finds an open terminal and uses it, either because she assumes the logged-in account on the terminal is hers or because she does not want to do the login step. Second, a malicious individual wants to use an open (logged-in) terminal while the already logged-in user is nearby, perhaps because the logged-in user has privileges desired by the adversary, or the adversary wants to take action in the name of the logged-in victim. This adversary may try to observe and mimic the logged-in user's hand movements to fool the terminal into falsely authenticating himself as that user.

IV. Approach

In this section we introduce bilateral authentication, compare it with traditional authentication methods, and give an

overview of ZEBRA.

A. Bilateral authentication

Traditional user-authentication schemes authenticate a user by comparing an attribute that the user produces with a previously stored attribute. For instance, password-based authentication schemes compare the hash of the user-entered password with the stored password hash, voice-based authentication schemes compare features of the user's voice against stored features of authorized users' voices, and a keystroke-based biometric scheme compares keystroke dynamics of the user with the user's stored keystroke dynamics.

In our case, bilateral authentication, the user is authenticated by comparing two observations of the same attribute of the user, measured separately in real-time by two sources (hence the term *bilateral*). A related example of a bilateral authentication method is the 'same-body authentication' solution by Cornelius et al. [24]. They measure a user's motion using multiple accelerometer sensors placed at different positions on the body and compare these sensors' signals (which measure the same attribute, the user's motion) to determine whether all the sensors are on the same body.

A general bilateral user-authentication method can be described as a user-authentication method where an attribute about the user, a, is observed and measured by two independent parties P and Q, where these measured signals could be the same (as in the above example) or different (as in ZEBRA) but the user is authenticated only if the two signals correlate. The attribute a could be physiological (e.g., heart rate, body temperature), behavioral (e.g., walking, user's interaction with device), or environmental (e.g., being in the same room, radio signal).

There are several benefits of bilateral authentication, some of which overlap with desired properties identified by previous research [16]. These include:

1) No need to store sensitive information in the authenticating device. Although sensitive information can be stored securely, in practice it is not stored securely, and when the system is compromised users' sensitive information is leaked [25]. By eliminating the need to store any sensitive information we eliminate this risk.
2) No mental burden on users. Users have to remember their password if they use a password-based authentication scheme; previous work has shown that users are not good at remembering passwords and they use work-arounds to avoid using passwords [4], [8]. In bilateral authentication there is no secret for users to remember.
3) No hassle for users over time. Users' habits and behaviors change over time, either naturally or due to an injury or illness. Behavior-based authentication schemes are susceptible to these changes, and they need to be re-trained for the user whose behavior changed over time. Bilateral authentication does not rely on any user-specific behavior, only on the fact that the user is doing specific interactions with the terminal.
4) No hardware modifications to the terminal, assuming it has short-range radio such as Bluetooth.

Bilateral authentication, however, cannot always be used because it is not always possible to monitor the users' interaction externally. Not all interactions between the user and desktop involve the dominant hand; notably, periods of screen reading involve no motion at all. We expect that, much as with common screen-saver software, users will need to periodically jiggle the mouse while reading extensively.

B. ZEBRA

ZEBRA provides continuous authentication, that is, it continuously verifies the identity of the logged-in user. Although continuous authentication has many uses, it is a necessary foundation for a smart deauthentication mechanism. Such a mechanism can automatically take protective action (such as locking the screen) when another user starts using a terminal that a previous user had logged in to.

ZEBRA works as follows: Jane, a ZEBRA user, logs in to the terminal, the terminal connects wirelessly to her bracelet (because she had paired them earlier, the terminal can look up her bracelet's network address, given her username, and seek that bracelet on the short-range radio connection). The presence of the bracelet may optionally be required by the login process, serving as a second-factor token, strengthening the initial authentication step. We are concerned here with what happens after login, continuously verifying that Jane remains the active user of the terminal. As Jane uses the terminal, the bracelet captures the sensor data (accelerometer and gyroscope) from Jane's dominant wrist movement and sends it to the terminal. From the acceleration data the terminal receives, it generates, using a classifier, a sequence of 'interactions' (mouse scrolling or typing) that Jane appears to be doing. The terminal also generates the actual sequence of interactions, based on the inputs it receives from keyboard and mouse. By comparing these two sequences of interactions, the terminal verifies whether the user using the terminal is Jane, i.e., the one wearing the bracelet.

When Jane steps away from the terminal and another user starts using the terminal, the two sequences of interactions will not match because the interaction sequence that Jane's bracelet generates when she is away from her terminal will be different from the interaction sequence of the other user on the terminal. Since the two interaction sequences will not match, the terminal will deauthenticate Jane and take action to prevent another user from misusing her account.

The idea of ZEBRA stems from two observations: *i)* people interact with most input devices with their hands, and *ii)* a user's hand movements when the user interacts with an input device can be correlated to the inputs the device receives. For example, when the user is scrolling or clicking, her fingers are moving but her wrist is relatively stationary; when the

user clicks the mouse and then starts typing on the keyboard (typically with both hands), her hand will move from the mouse to keyboard. Thus, the hypothesis driving ZEBRA is that if we can capture the user's hand movement and compare it with the inputs the terminal receives, we can determine whether the user is using the terminal.

Figure 1 shows a user's wrist acceleration when she was interacting with the terminal. The x-axis represents the time (in seconds) from the start of the experiment and the y-axis represents the magnitude of the acceleration, as measured by the bracelet on the wrist. We marked, with shaded regions, three types of user interactions in the figure: *scrolling*, *typing*, and *MKKM*, where MKKM stands for 'Mouse to Keyboard or Keyboard to Mouse' interaction representing the action of switching between keyboard and mouse. As shown in the figure, the user scrolled the mouse at 65.5 s, from 66.3 s to 74.4 s, and then briefly at 75.1 s. The graph shows that her wrist was relatively still during scrolling, as one would expect. When she moved her hand from mouse to keyboard (around 77 s) to type, we see a sudden spike in acceleration caused as she lifted her hand off the mouse and as she rested her hands on the keyboard. As she typed (77.5 s to 83.4 s), we see small changes in the acceleration, implying that her wrist moves little during typing. After typing she switched from keyboard to mouse (around 83.5 s), and we see another sudden spike in the acceleration.

We can see the differences in the acceleration patterns between interactions. For instance, broadly speaking, there is more wrist movement during typing than scrolling, but less than when she switches between keyboard and mouse. This example supports our hypothesis that we can generate a sequence of interactions from a user's wrist movement.

The acceleration data that is not marked in the graph represents the user's other interactions with the terminal such as mouse-dragging, clicking, or hand movements not involving interaction with the terminal; we highlighted only three types of interactions on the graph for readability.

C. Dealing with adversaries

ZEBRA deals with the two adversaries described in Section III as follows:

In the case of the innocent authorized user who wants to use an already open terminal for her own task, if Sally attempts to use the terminal that Tina left open, the terminal will try to verify whether the current user (Sally) is Tina. If the terminal does not receive data from Tina's bracelet, e.g., because Tina is not near the terminal, ZEBRA will log Tina out and will attempt to log Sally in. If Tina is near the terminal but not using the terminal, e.g., she may be talking to a nurse, then ZEBRA will attempt to correlate Tina's bracelet movements with Sally's inputs on the terminal. The classification will fail with high probability, and ZEBRA will log Tina out and attempt to log Sally in. Thus, ZEBRA will prevent an innocent authorized user from performing a task, e.g., updating a patient's record, with another authorized user's credentials.

ZEBRA deals with the case of a malicious individual in a similar fashion. In the second use-case, Claire leaves her terminal unattended, and Jake manages to get access to her terminal before the terminal times out and auto-locks. As Jake tries to navigate the terminal using the keyboard and mouse, the terminal will try to correlate Jake's inputs with Claire's hand movements. Assuming that it is hard to control a terminal at will by mimicking Claire's hand movements while she is around the corner talking on the phone, the correlation will fail in a similar manner to the previous case and, therefore, the terminal will lock, preventing Jake from misusing Claire's account. Our evaluation shows that our assumption is reasonable in the case of humans trying to mimic human hand movements. We touch on the resilience of ZEBRA to automated attacks in Section VII-C.

V. Method

In this section, we describe the ZEBRA architecture and our approach to correlate a terminal's input with bracelet acceleration data.

A. Architecture

Figure 2 shows the architecture of ZEBRA. As shown in the figure, there are five main components in ZEBRA. The *interaction extractor* extracts interactions from a user's keyboard and mouse inputs, and sends the sequence of interactions to the authenticator and the time intervals of the interactions to the segmenter. The *segmenter* segments the accelerometer and gyroscope data into blocks based on the time intervals it receives from the interaction extractor. The *feature extractor* extracts features for each block of data that it receives from the segmenter. The *interaction classifier* takes these features and classifies them into one of our specified interactions. The *authenticator* compares the actual sequence of interactions that it receives from the interaction extractor and the inferred sequence of interactions that it receives from the interaction classifier, and it makes a decision whether the two users – the current terminal user and the user wearing the bracelet – are the same or different. If they are different then we need to deauthenticate the bracelet user, who is currently logged in to the terminal. Based on the system policy or user preference, ZEBRA can either logout the user, lock the screen, raise an alarm, or take some other action. Below we discuss each component in detail.

B. Interaction extractor

As mentioned, this component extracts 'interactions' from the input events stream generated by the OS when the user provides inputs to the terminal via keyboard or mouse. We use three main types of user interactions with a terminal: *MKKM*, *scrolling*, and *typing*. There are other interactions, such as moving the mouse, dragging the mouse, or clicking

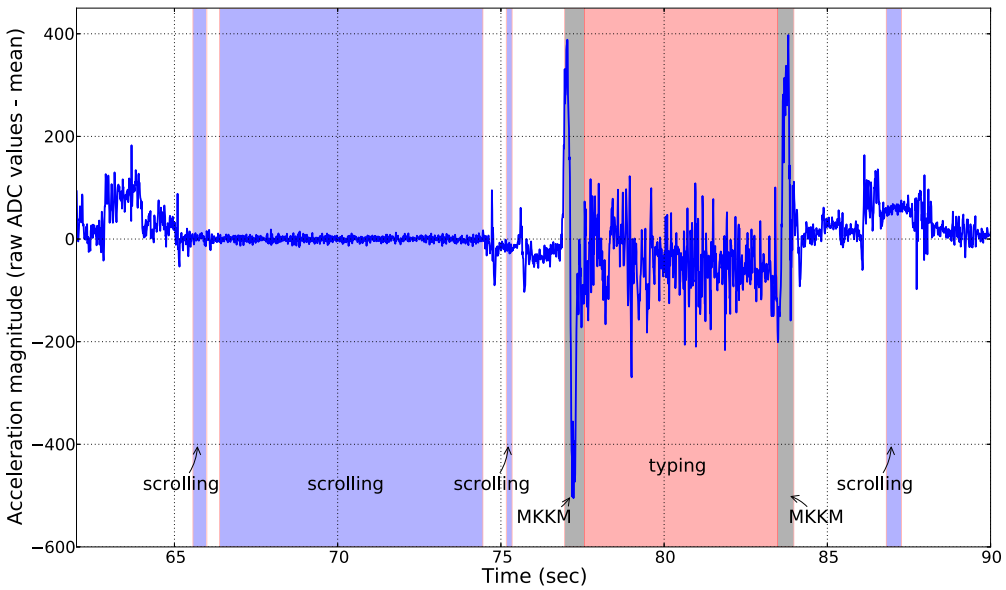

Figure 1: Acceleration of user's wrist when she is using a computer terminal.

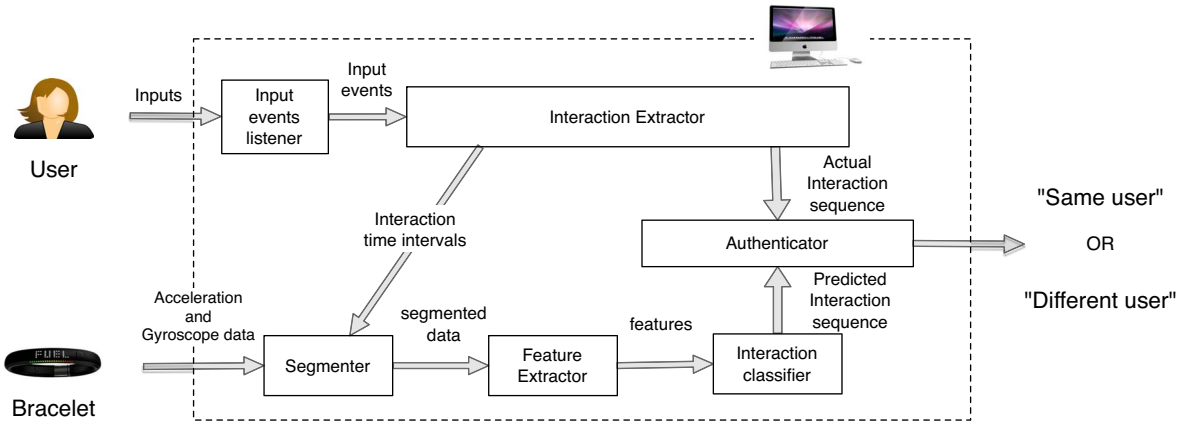

Figure 2: ZEBRA architecture.

the mouse, but we do not consider them because in our evaluation they did not contribute to ZEBRA's performance.

MKKM. This interaction captures the users' dominant hand (here, we mean the mouse hand) movement when she switches from the mouse to the keyboard or from the keyboard to the mouse; *MKKM* is short for 'Mouse to Keyboard or Keyboard to Mouse'. An *MKKM* interaction consists of a mouse-related event followed by a keypress event or vice-versa.

There is, however, a challenge in identifying whether the keypress event followed by a mouse-related event was caused by the dominant hand or the other hand, because the user may press a key with her non-dominant hand while keeping her dominant hand on the mouse. With one bracelet, we cannot identify such events with certainty. We account for such events by dividing the keys on the keyboard into three zones, depending on which hand the user is likely to use to press that key: left zone, middle (or ambiguous) zone, and right zone, as shown in Figure 3. We introduced the 'middle' zone because not everyone types according to two-handed typing guidelines, which divides the keyboard into two zones, and we noticed some subjects used either hand to type the keys in the middle zone. So, if the user is right handed (that is, uses the mouse with her right hand) and presses a key in

the right zone after a mouse event, we assume she moved her dominant hand. Thus, we assume that users will stick with our zone divisions, i.e., use their left hand for keys in the left zone and their right hand for keys in the right zone. Some users may break this assumption, but this heuristic seemed to work well, because to identify MKKM we only need the user to press any one key in the right (or left) zone with their right (or left) hand, and we observed that all our subjects did use two hands when typing.

Scrolling. This interaction captures users' use of a scroll-wheel built-in to the mouse. When a user is scrolling, `ScrollWheel` events are continuously generated by the OS, each event reporting the amount of scroll performed by the user since the last scroll, so that the application can update the UI accordingly. We define a *scrolling* interaction as a sequence of uninterrupted `ScrollWheel` events.

However, sometimes the mouse is slightly moved because the user's hand is not still, and we observe some `MouseMove` events in the `ScrollWheel` events stream. The idea behind this interaction is to capture the durations during which the user was using the mouse and her hand (wrist) was relatively stationary, so we ignore small mouse movements. We consider a mouse movement as small if the associated `MouseMove` events in the `ScrollWheel` events stream are few in number (e.g., 5 events) and the cumulative mouse displacement indicated by these `MouseMove` events is small (e.g., 5 pixels). These thresholds (minimum number of `MouseMove` events and maximum mouse displacement) are parameters in our experiments.

Thus, we define a *scrolling* interaction as a sequence of `ScrollWheel` events with few intervening `MouseMove` events such that the total mouse displacement is small (below a certain threshold).

Typing. This interaction captures the users' use of the keyboard. When a user hits a key, she first presses the key down, and then as she removes her finger she releases the key up. Associated to these actions, two events are generated by the OS for each keypress: `KeyDown` and `KeyUp`. Thus, we define a *typing* interaction as a sequence of `KeyDown` and `KeyUp` events.

If there is a continuous keypress events stream with mouse-related events in between, we count those keypress events as separate typing interactions, separated by the mouse-related events. Unlike scrolling, where we ignored small numbers of mouse-related events, during typing any mouse-related event means that the user moved her hand from keyboard to mouse, which is an MKKM interaction. Thus, for a keypress events sequence with few mouse-related events in between, we extract at least four interactions: typing, MKKM (to switch to mouse), MKKM (to switch back to keyboard), typing, and maybe scrolling between the two MKKM events if the user scrolled the mouse-wheel.

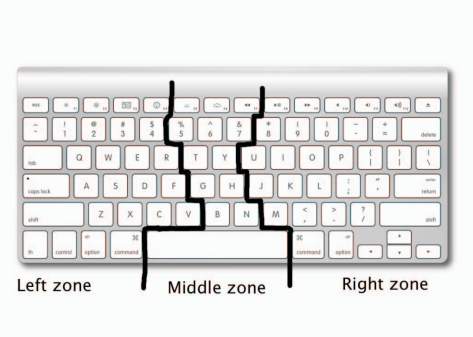

Figure 3: Keyboard divided into left, middle, and right zones.

Extraction. When extracting interactions from input events, we apply three constraints: *idle threshold*, *minimum duration*, and *maximum duration*. Idle threshold is the maximum time difference between two consecutive events in an interaction. The rationale behind this constraint is to capture only the interactions that involve the user's continuous interaction with the terminal and eliminate interactions during which the user does tasks other than using the mouse and the keyboard. During a pause, there is no input to the terminal; we do not know what the user is doing, and thus cannot correlate with the user's wrist movement. If there is a pause greater than the threshold, we split the interaction into two interactions separated by the pause. For example, if in a series of keypress events there is a 2 min pause, then we split these keypress events into two *typing* interactions, one before the pause begins and one after the pause ends.

The other constraints refer to the minimum and maximum duration of interactions. If an interaction lasts for less than the minimum duration, we ignore it, and if an interaction exceeds the maximum duration we split it into two consecutive interactions. While splitting the interaction we do ensure that the new interaction is longer than the minimum duration: if the new split interaction has duration less than the minimum duration, we do not split the interaction; thus, we can have interactions that are almost as long as *minimum duration* + *maximum duration*.

Based on these three constraints and the definitions of the interactions described above, this component outputs a sequence of interactions from given input events. This sequence of interactions, IE, is of the form

$$(I_0, t_0, t_1), (I_1, t_2, t_3), \ldots$$

where I_0 is an interaction ID (corresponding to one of the three described interactions) that starts at time t_0 and ends at t_1, and similarly interaction I_1 spans (t_2, t_3).

From the sequence IE, interaction ID sequence $(I_0, I_1, \ldots)$ is sent to the authenticator. The interaction timings sequence $((t_0, t_1), (t_2, t_3), \ldots)$ is sent to the segmenter.

C. Segmenter

This component receives accelerometer and gyroscope data from the user's bracelet. The accelerometer data is of the form

$$(t_i, x_i, y_i, z_i), (t_j, x_j, y_j, z_j), \ldots$$

where (t_i, x_i, y_i, z_i) represents one acceleration data sample taken at time t_i and the instantaneous accelerations along x, y, and z axes are x_i, y_i, z_i, respectively. The gyroscope data is of the similar form

$$(t_i, a_i, b_i, c_i), (t_j, a_j, b_j, c_j), \ldots$$

where (t_i, a_i, b_i, c_i) represents one gyroscope data sample taken at time t_i and represents the instantaneous rotational velocity along x, y, and z axes, a_i, b_i, c_i, respectively.

As shown in Figure 2 the segmenter receives actual interaction time intervals from the interaction extractor. The segmenter breaks the acceleration data stream into blocks corresponding to each time interval, using the time of each data sample and the time of the intervals. For the time-interval sequence, $((t_0, t_1), (t_2, t_3), \ldots)$ this component will place all the accelerometer and gyroscope data samples with time $t_0 \leq t \leq t_1$ into the first block, all the data samples with time $t_2 \leq t \leq t_3$ into the second block, and so on. These data blocks are sent to the feature extractor. Acceleration and gyroscope samples outside interaction intervals are discarded.

In most signal-processing algorithms, data is segmented into blocks (also called *windows*) of equal size, but in our case the block sizes are variable. There are two main reasons to perform segmentation this way. First, when the user is not interacting with the terminal, we do not have any interaction sequences to use for authentication, so we ignore the sensor data for durations when she is not interacting with the terminal. Second, the user's interactions themselves are of variable duration so it makes sense to chunk accelerometer data this way. For the durations when she is interacting, one could segment sensor data into blocks of equal size and infer an interaction for each block, but given that a user's interactions are of variable duration, it is likely that one sensor data block would contain data for one or more interactions, which would reduce the classifier performance. Variable segmentation ensures that each sensor data block contains data for just one interaction.

D. Features

This component receives sensor data in blocks, and it computes a feature vector over each block. We do not know the orientation of the user's bracelet, so we ignore the orientation (individual axis accelerations and angular velocities) and just use the magnitude of acceleration and angular velocity. For each acceleration data sample (t, x, y, z), the magnitude m is given by

$$m = \sqrt{x^2 + y^2 + z^2}$$

and for each gyroscope data sample (t, a, b, c), the magnitude r is given by

$$r = \sqrt{a^2 + b^2 + c^2}.$$

After computing these magnitudes, we now have for each block a series of magnitudes $(m_0, r_0), (m_1, r_1), \ldots$.

We compute the following 12 features over each series of acceleration and angular velocity magnitudes in a segmented interaction block: *mean, median, variance, standard deviation, median absolute deviation (MAD), inter-quartile range (IQR), power, energy, peak-to-peak amplitude, auto-correlation, kurtosis*, and *skew*. We chose the first seven features because others have used them successfully for activity recognition [26] and for correlation among different accelerometer signals [24]. We add the latter five features to capture the patterns of the three interactions that we noticed. During MKKM, there is a sudden spike in positive and sometimes in negative direction, so we use *peak-to-peak amplitude*. Because the placement of the peaks in a MKKM is towards the start of the interactions, we use *skew* as a feature. During typing, the peakedness is distinct, and so we included *kurtosis* as one of our features. The wrist movement pattern during a typing or scrolling interaction should be roughly similar, unlike MKKM, so we use the *auto-correlation* feature to capture that difference.

For each block of data, we compute a feature vector $F = (f_0, \ldots, f_{11})$, and send the sequence of feature vectors $F_0, F_1, \ldots$ (each corresponding to one interaction block) to the interaction classifier.

E. Interaction classifier

The classifier takes a feature vector F as input and outputs an interaction ID, its inference that the sensor data associated with that feature vector represents that interaction.

To train the classifier, we segment a subject's wrist sensor data based on her actual interaction timings, as described above. We feed the classifier with feature vectors corresponding to the actual interaction and provide the actual interaction labels. Later, when evaluating our approach with a given subject, we use a classifier that was trained with other subjects' data, because our intent is for the classifier to be user agnostic.

The classifier receives a sequence of feature vectors and it outputs its inference, a sequence of interaction IDs $(i_0, i_1, \ldots)$. It then sends this sequence to the authenticator.

We explored two classifiers: Naive Bayes classifier and Random Forest classifier. For our dataset, the Random Forest classifier outperformed the Naive Bayes classifier; the results reported in Section VI are with the Random Forest classifier.

F. Authenticator

The authenticator matches two sequences: the sequence of actual interactions and the sequence of interactions inferred by the classifier based on the user's wrist movement. If the

two sequences match, the authenticator outputs 1 indicating that the current terminal and the bracelet user are the same. On the other hand, if the two sequences do not match, it outputs 0 indicating that the two users are different. If the users are different, we need to deauthenticate the bracelet user, who is logged in the terminal.

To match the two sequences, we use four parameters: window size, overlap fraction, match threshold, and grace period. The window size, w, is the number of interactions the authenticator compares at a time. The overlap fraction, f ($0 \leq f < 1$), indicates how much we should overlap the moving window, 0 being no overlap. For each window the authenticator computes a matching score (between 0 and 1) indicating how well the two sequences match in that window; 0 being no match at all, and 1 being a complete match. If the matching score for a window is greater than the match threshold, m, we output 1 for that window, indicating that the terminal user and the bracelet users are the same for that window. Otherwise, we output 0 for that window.

If we incorrectly output 0 for a window and deauthenticate the user immediately, it would frustrate the user. To account for such false negatives, we introduce the grace period parameter, g. This parameter indicates how many consecutive window scores of 0 are measured to deauthenticate the user. For example, if $g = 3$ then we should get 0 for three consecutive windows before we deauthenticate the user. We reset the zero-count when we get a window with output 1. This parameter increases convenience but also increases security risk; we keep its default value low.

VI. Evaluation

As mentioned in Section III, we desire ZEBRA to be continuous, passive, unobtrusive, user-agnostic, quick, and accurate. We achieve the first four properties by design. ZEBRA requires no explicit input from the user and as long as the user is in (radio) proximity, i.e., the user's bracelet can send data to the terminal, ZEBRA continuously verifies the presence of the user; thus, ZEBRA does continuous authentication passively. The bracelet can potentially monitor a user's physical activity, which may be sensitive information for some users. ZEBRA respects users' privacy, and it does not monitor the user's movements when the user (and no one else) is not using her logged-in terminal. While evaluating ZEBRA for a user, we did not train the classifier using that user's data. Hence, ZEBRA is user-agnostic, i.e., independent of the user's behavior when she is using a terminal. We evaluate accuracy and quickness through a user study and present the results in this section.

A. User study

We recruited 20 subjects for our user study, using flyers posted across our college campus and online. Table II shows demographic data about the subjects. Subjects took about 30 to 40 mins to complete the user study; they received $10 as compensation. Our research protocol was approved by our campus IRB.

Table II: Demographics of user study participants.

	Category	# of subjects
Gender	Male	7
	Female	13
Field	Computer Science	8
	Non-CS	12
Age	18-25	15
	25-30	5
Handedness	Right	19
	Left	1

The user study consisted of three experiments. The first experiment was designed to capture the users' hand movements as they interact with a desktop in *normal* use. Subjects were instructed to imagine that they were in a public cafe and were asked to browse the web for 10 min. They were told that everything they typed would be logged and so were asked not to enter any sensitive information. Further, they were asked not to read any long articles or watch videos, as it would not provide much data for our study.

We designed our second experiment to collect user interaction data in a more controlled setting. Users were asked to fill out a small web form, which required users to type, scroll, drag the mouse, click, and move the mouse; they were asked to fill this web form five times.

Our third experiment was designed to collect data to test a malicious adversarial case. For this experiment, we asked each subject to be a malicious adversary whose goal was to mimic the victim user's mouse-hand movements to the best of their abilities. The victim user (one of the researchers) filled out the same web form that the subjects used in Experiment 2; thus, the subjects were already accustomed to the task. We realize that a real adversary can be motivated and skilled enough to mimic users very well, compared to our subjects. So we decided to assist the subjects when they were performing the role of a malicious adversary. To assist them in mimicking the victim, we made sure they had a good view of the screen and the victim's hand movement, we increased the cursor size, and the victim user gave verbal clues before he began an action. For example, the victim would say 'typing' before he began typing. He would say '2' when he was going to fill the question number 2 in the web form. The victim tried to use the same pace to fill out the web form for all subjects, but reduced the pace for some subjects when they were lagging too far behind. It should be noted that this experiment was intended to be favorable for the adversary.

Each subject performed, on average, about 192 Scrolling interactions, 293 Typing interactions, and 146 MKKM interactions in the three experiments. After these three experiments, we asked each subject to walk for a few minutes and to write on a paper, so we could collect data for walking

and writing activities, because these are common activities for a user that steps away from the terminal. We use this data to evaluate how quickly ZEBRA can deauthenticate a user when she does one of these tasks while another user attempts to use her terminal.

B. Data collection

In our user study, we collected two types of data about subjects' interaction with the terminal: i) inputs received by the OS through keyboard and mouse, captured by a script we wrote; and ii) the subject's hand movement, captured by a sensor worn by the subject on her dominant wrist.

We used iMacs for our subject study. Subjects used an iMac with an Apple keyboard and Apple mouse with a scroll ball. We wrote a Python script that uses Apple's Cocoa APIs for OS X and captures all keyboard and mouse input events generated by the operating system when the subject provides input using those input devices. The captured input events were `KeyDown`, `KeyUp`, `MouseMove`, `ScrollWheel`, `LeftMouseDown`, `LeftMouseUp`, and `LeftMouseDragged`, which are generated when the subject presses and releases a key, moves the mouse, uses the scroll-wheel, presses and releases the left mouse button (left click), and drags the mouse, respectively.[1] For each keypress event the OS reports the time when it is pressed/released, the key value, whether the subject is holding the key down, and whether the subject is repeatedly pressing the key. For mouse-related events, the OS reports the time the event was generated, absolute coordinates of the mouse pointer on the screen, pointer displacement (in pixels) since the last mouse event, and scroll length (i.e., how much the subject scrolled the wheel). We log all this information, but in the current implementation of ZEBRA, we use only the time of event, event type, key value for keypress events, scroll duration, absolute mouse pointer coordinates, and mouse pointer displacement.

We used a Shimmer [27] to capture the subjects' hand movements, by asking each subject to wear a Shimmer device on the wrist of the hand they normally use to operate a mouse. The Shimmer contains an accelerometer sensor, a gyroscope sensor, and a Bluetooth radio. Once connected to the terminal over Bluetooth, the Shimmer streams its accelerometer and gyroscope readings to the terminal at 500 Hz, where they are logged to a file. We calibrated all Shimmers' accelerometers and gyroscopes prior to their use.

C. Results

In this section we evaluate ZEBRA's accuracy and how quickly it deauthenticates users when an adversary starts using their logged-in terminal.

[1]Although our implementation is for MacOS X, the same kinds of information are available in Windows and Linux so our method should be easily portable to other systems.

1) Accuracy: We use two metrics to evaluate the accuracy of ZEBRA. The false-positive rate (FPR) is the fraction of the testing data that is negative but misclassified as positive; in ZEBRA the FPR is the fraction of all interactions where an unauthorized user is authenticated as an authorized user. Similarly, the false-negative rate (FNR) is the fraction of all interactions where an authorized user is misclassified as an unauthorized user.

A high FNR indicates that the system classifies an authorized user incorrectly as an unauthorized user more often. When this happens, the system takes protective action, such as logging out the user or locking the terminal; both actions will annoy the authorized user using the terminal. A negative result indicates to the system that an unauthorized individual is using the terminal and the system takes action, such as logging out the user or locking the terminal. A **false** negative is, as one might imagine, annoying to an authorized user of the terminal. Thus, from the usability point of view a low FNR is desirable. Figure 4 shows the average false-negative rate across all subjects for different window sizes and thresholds.

As described above, ZEBRA classifies the terminal user as the bracelet user by comparing the actual interaction sequence and the interaction sequence inferred by the classifier from bracelet data. The comparison is performed over a given window size. Thus, we compute an FNR as the fraction of all windows where ZEBRA misclassified the authorized user as an unauthorized user. Then each data point in the Figure 4 is the average of the FNR across all subjects for a given window size and threshold value.

The threshold parameter indicates the fraction of interactions in a window that should match for ZEBRA to consider the user as the authorized user. Thus, the threshold value indicates how strict ZEBRA is when correlating interactions, and as expected the FNR is smaller for smaller threshold values and it increases with threshold values. Window size is the number of interactions matched at a time to authenticate the user; a larger window size allows more interactions to be matched, so the FNR drops as the window size increases. ZEBRA performs best in terms of authenticating a user for window sizes greater than 20; thus, the more interactions a user provides while working on the terminal, the better ZEBRA performs. In terms of threshold values ZEBRA provides best FNR for 0.5 and 0.55, and reasonably well for threshold value of 0.6. A low threshold value improves usability but, as we show below, it reduces security, so we need to choose the trade-off carefully.

For a given subject and adversary, the FPR is the fraction of all windows where ZEBRA misclassified the adversary as the subject (authorized user). We compute a FPR for each subject as the average FPR across all adversaries; each data point in the following FPR graphs is the average of these FPR across all subjects. A high FPR indicates that we falsely authenticate an unauthorized user as the logged-in

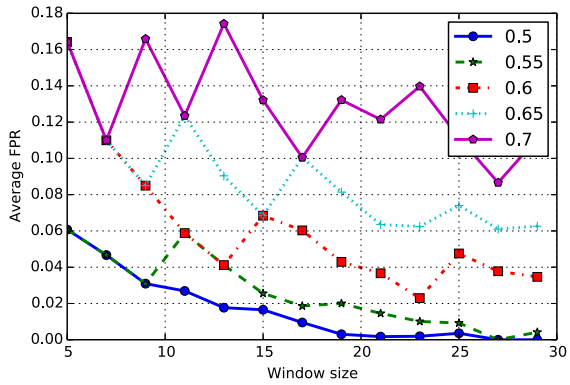

Figure 4: Average false-negative rate vs. window size for different thresholds (0.5, 0.55, 0.6, 0.65 and 0.7). ZEBRA performs best in continuously authenticating users for window sizes larger than 20.

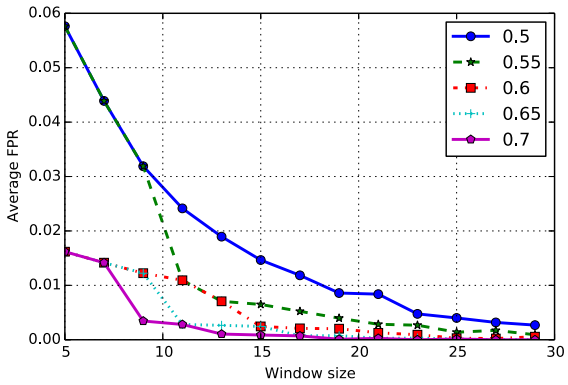

Figure 5: Average false-positive rate when the adversary is accessing the terminal while the logged-in user is walking nearby.

user, allowing him to access the logged-in user's account, which is undesirable. Thus, a low FPR is good from a security point of view. Window size is the number of interactions that ZEBRA is allowed to consider to issue the decision whether the current user is the same as the logged-in user. Thus, ideally we want a low FPR for a small window size.

Figures 5 and 6 show the average false-positive rate when the adversary is using the terminal and the logged-in user is walking and writing, respectively, near the terminal. (Note the different y-axis scales.) As expected, the FPR is smaller for the higher thresholds. The FPR is low and drops quickly with respect to window size, when the user is walking compared to when she is writing because the wrist movements while walking are very different to wrist movements when a user is using the terminal, whereas the wrist movements during writing are somewhat similar to terminal use. Figure 5 shows that the FPR is below 0.02 for thresholds 0.6 and above, even for short window sizes. The FPR in the user-writing case drops below 0.03 for threshold 0.6 for windows of size 15 or greater. Thus, ZEBRA performs reasonably well even if the user is performing an activity that is somewhat similar to working on a terminal in terms of hand movements.

In our third experiment, we imagine a malicious adversary: the user is logged-in on a terminal A but steps aside to work on a nearby terminal B, and the adversary starts using terminal A while trying to mimic the user's hand movements and similar interactions. If the adversary succeeds in mimicking the user's hand movements while providing similar interactions to terminal A, then ZEBRA will misclassify the adversary as the user and the adversary can continue using the terminal. In our experiment we asked the subjects to be the malicious adversary and try to mimic a user (a researcher). Both the subject and researcher performed the same tasks (filling web forms), and the researcher's screen

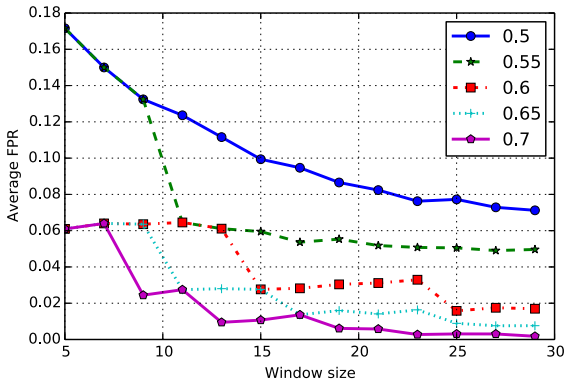

Figure 6: Average false-positive rate when the adversary is accessing the terminal while the logged-in user is writing nearby.

and hands were clearly visible to the subject. Figure 7 shows the false-positive rate for this case. The FPR rate drops below 0.04 for windows of size 15, and threshold 0.6 and above. Thus, even when the adversary and the user were performing the same task on nearby terminals, and the adversary was trying to mimic the user's actions, ZEBRA performed well in recognizing the adversary. Thus, ZEBRA should be able to recognize a change in user even in an environment where the previous user is working on a nearby terminal when a new user walks to the unlocked terminal to use it.

In Table III we show the mean and standard-deviation of FNR and FPR for all subjects for threshold of 0.6 and four window sizes. The high variability in FNR is because of Subject 1, whose wrist movement during keyboard and mouse interaction were very different compared to the other subjects. To keep ZEBRA user-agnostic, for each subject we train the classifier using other subjects' data, but if a subject's

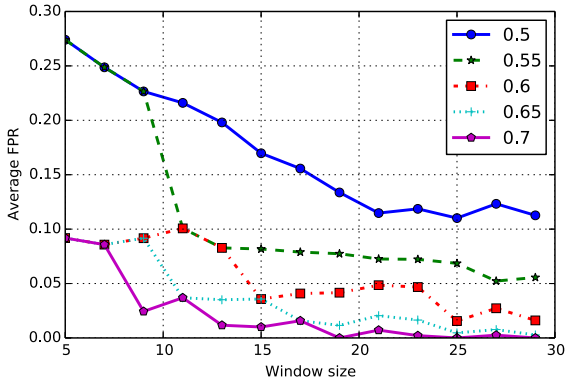

Figure 7: Average false-positive rate when the adversary is trying to access the user's logged-in terminal by mimicking the user who is using a nearby terminal.

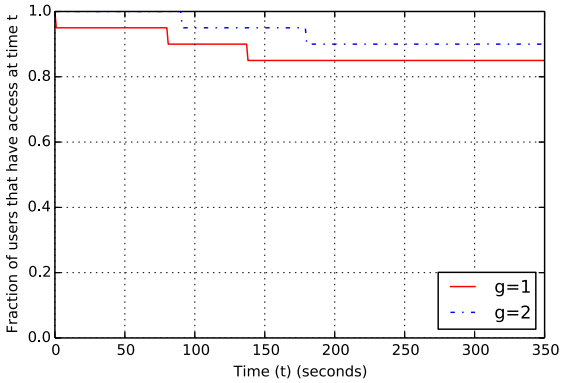

Figure 8: Fraction of authorized users that have access to the terminal at time t, for optimal window size = 21 and optimal threshold = 0.6.

interaction is very different than other subjects, the classifier cannot accurately classify that subject's interactions, which affects ZEBRA's accuracy in verifying that subject. This can be resolved by training the classifier on a larger population or training the classifier for these specific subjects. If we exclude Subject 1, we get low variability and even better FNR, as shown in the last column (FNR[3]) in Table III.

From the FPR and FNR results we found the parameters that give a reasonable tradeoff between usability and security with ZEBRA are window size of 21 and threshold of 0.6. For window size of 21, ZEBRA verifies the user after every 21 interactions, which can take at most 21 s if the user is providing inputs continuously, because each interaction is at most 1 s long. However, in our experiment subjects took about 6 s for 21 interactions. We use these optimal parameters to evaluate *quickness* of ZEBRA in terms of the time ZEBRA takes to recognize an unauthorized user.

Table III: Average FNR and FPR for different Window sizes (W). Mean (and standard-deviation) of all subjects.

W	FNR	FPR[1]	FPR[2]	FNR[3]
5	0.164 (0.155)	0.016 (0.012)	0.061 (0.064)	0.140 (0.118)
13	0.041 (0.077)	0.007 (0.008)	0.061 (0.088)	0.026 (0.038)
21	0.037 (0.096)	0.001 (0.002)	0.031 (0.057)	0.017 (0.044)
29	0.035 (0.094)	0.001 (0.001)	0.017 (0.035)	0.015 (0.034)

[1]The adversary is using the terminal while the user is walking nearby.
[2]The adversary is using the terminal while the user is writing nearby.
[3]Mean (and standard-deviation) FNR for all subjects, excluding Subject 1.

2) Quickness: When the user changes (i.e., when the current user is different than the logged-in user), we want to identify the change immediately so we can prevent any accidental or intentional misuse of the logged-in user's account. We define *quickness* as how 'soon' we can identify a changed user, where 'soon' can be measured in time or windows, where a window represents a fixed number of interactions. We use the 'duration of attack success' as a metric to evaluate how quickly ZEBRA detects an adversary and 'duration of inappropriate lockout' as a metric to evaluate how often ZEBRA will lock out an authorized user because it misclassified the user as an adversary. A smaller duration of attack success is better as it gives a smaller attack window for the adversary. On the other hand, it is desirable to have extended periods of time without inappropriate lockouts in order to improve usability.

Figure 8 shows the fraction of users that are recognized as authorized users by ZEBRA at time t for a grace period (g) of 1 and 2 windows. This figure shows the first instance in time when ZEBRA misclassifies the user as an adversary *and* takes action according to the system/user policy, which can be to lock the terminal or log out the user. For instance, 95 % of users were still recognized as authorized users at 50 s, or said another way, 5 % of users were misclassified by ZEBRA by time 50 s and may be required to re-authenticate themselves. For grace period of 1 window, ZEBRA correctly recognized 85 % of users throughout their session on the terminal. We can improve this number by increasing the grace period. As shown in the figure, for grace period of 2 windows, ZEBRA recognizes 90 % of users correctly throughout their use of the terminal.

Figure 9 shows the fraction of adversaries that are recognized as authorized users by ZEBRA at time t for grace periods (g) of 1 and 2. This graph shows how quickly ZEBRA can recognize an adversary and terminate his access to the logged-in user's account on the terminal. As shown in the figure, for grace period of 1 window at time $t = 0$, all adversaries have access to the terminal, but within 5 s only 40 % of adversaries have access – ZEBRA identified on average 60 % of adversaries as unauthorized users within 5 s, and by $t = 11$ s, ZEBRA identified all

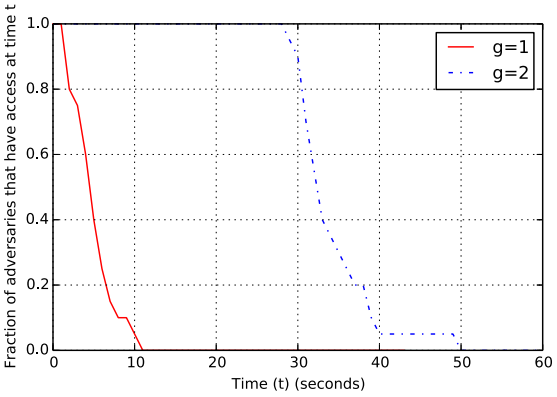

Figure 9: Fraction of adversaries that have access to the terminal at time t, for optimal window size = 21 and optimal threshold = 0.6.

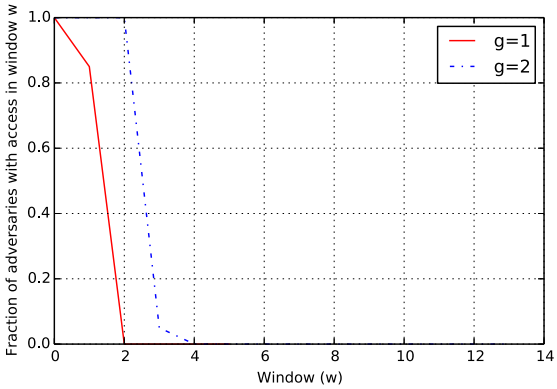

Figure 10: Fraction of adversaries that have access to the terminal at the end of window w, for optimal window size = 21 and optimal threshold = 0.6.

adversaries. A grace period of 2 windows improves usability of ZEBRA, as shown in Figure 8, but it also increases the attack duration for adversaries; nonetheless ZEBRA still identified all adversaries within 50 s, much faster than typical deauthentication timer methods.

Figure 10 represents the above graph but in terms of windows instead of time, i.e., the fraction of the adversaries that have access to the terminal at the end of window w, where window size is 21 and threshold is 0.6. ZEBRA identified all adversaries for grace periods of 1 and 2 by the end of window 2 and 4 respectively. As we mentioned above, the window size is determined by the number of interactions (in this case 21), but interactions have variable duration; to compare with the previous figure, for all adversaries a duration of 2 windows was approximately 11 s.

VII. Discussion

ZEBRA allows the adversary a small attack window before it can identify him as an unauthorized user; the adversary can misuse the logged-in user's account within this window. This window arises because ZEBRA identifies an unauthorized user based on that user's inputs, and it needs to collect enough inputs to make a decision with high probability. This attack window exists for all passive continuous authentication schemes that leverage user inputs for authentication, including keystroke biometrics and mouse biometrics. The effects of this attack window could be reduced with some help from the operating system (OS), if the OS buffers the user's inputs and actions, and has the ability to roll-back actions whenever the inputs fail to authenticate the user. Nonetheless, without ZEBRA, an adversary will have unrestricted access to the terminal until he is caught, whereas with ZEBRA, he has unrestricted access for only a short duration.

Sometimes there may be two users giving input to a terminal, e.g., two users working together or a clinician asking for IT help from a staff. If the two users are giving input one at a time, then ZEBRA can be easily configured to authenticate both users for that terminal. However, if the users are giving input at the same time, e.g., one user is typing and another is handling the mouse, ZEBRA currently cannot verify either user; the user could temporarily disable ZEBRA to allow access to both users.

A. Deauthentication response

At its heart, ZEBRA is a method for continuous authentication. In this paper, we motivate the need for continuous authentication as a tool for automating deauthentication when a new person tries to use a terminal that has another another user logged-in. We emphasize, however, that the desired response to such situations is a matter of policy. Once ZEBRA decides that the terminal is now being used by someone other than the logged-in user, the policy might dictate that it lock the screen, or that it log-out the current user. With small modifications, a graduated response is possible. Recall that ZEBRA uses a threshold parameter m and a grace-period parameter g in making its decision; instead of outputting a binary decision, we could arrange for ZEBRA to output a probability intended to indicate its *confidence* that the user input is coming from the logged-in user. As long as the probability remains high, the terminal operates normally. When the probability drops, the screen may dim, then darken, then lock, then log out – in each case offering the user an opportunity to take an action (increasingly complex, as the confidence gets lower) to restore confidence in her authenticity. Such an approach can improve usability without necessarily lowering security and is a topic for future work.

B. Application to initial authentication

We describe ZEBRA as a complement to initial authentication methods such as passwords, biometrics or hardware

tokens [28]. If the ZEBRA bracelet (token) can be strongly tied to a specific user, such as through wrist biometrics [22], it may be possible for ZEBRA itself (with major modifications) to be used for initial authentication. Since ZEBRA authenticates users based on their inputs from the keyboard and mouse, a ZEBRA-based initial authentication method would involve tasks the user has to perform as part of the initial authentication, and these tasks will generate enough inputs for authentication. For example, as part of initial authentication, the user could be asked to type a displayed text, draw a circle, or scroll through a window. There exist authentication methods that require the user to perform tasks such as choosing a specific face among a group of faces. The benefit of a ZEBRA-based method would be that it does not place any mental burden on the user to remember any secret – users do the tasks displayed on the terminal. The challenge would be to design initial authentication that is short but generates enough user inputs to achieve high confidence.

C. Automated attacks

We designed experiment 3 to explore the extent to which an adversary may be able to defeat ZEBRA in an open terminal, by observing the wearer of the bracelet that opened the terminal. Our results suggest that it is hard to use an open terminal while mimicking an individual.

In principle, it is possible to automate the observation and monitor the victim using a video camera in combination with the use of special hardware (keyboard and mouse) that would release keystrokes and mouse input to the system at the specific times when the camera observes motion that may be consistent with a keyboard, MKKM, or scrolling interaction. In this way, the attacker would only have to worry about operating the terminal without having to consciously observe the victim. Our view is that this kind of sophisticated attack is stronger than what is necessary to beat passwords today. A video camera can be used to obtain passwords [29]. Also, if you can plug a custom set of keyboard and mouse into a terminal, you can potentially plug a hardware key logger (between the keyboard and the system) as well [30].

D. Extension

In this work, we focused on the deauthentication problem for desktop computers because we were motivated by associated problems faced by healthcare professionals in hospitals. It would be natural to extend ZEBRA to mobile devices, such as smartphones or tablet computers, and we believe this is possible. However, we should note that mobile devices present different challenges. First, users move their wrists less when interacting with mobile devices (e.g., while using mobile phones users often move only their fingers) compared to when they interact with computer terminals. Second, users could be on the move when interacting with mobile devices, whereas they are not moving when using a terminal. Third, the solution would have to be extremely energy-efficient to be able to run continuously on mobile devices. On the other hand, mobile devices do present some opportunities. For example, most mobile devices have built-in motion sensors, which could be leveraged for authentication.

ZEBRA could also be extended to other devices such as TV-remotes, game controllers, medical devices – any device where the user provides frequent inputs with her hand. For these devices, however, the application may be more for improving usability than security. For example, if the TV remote could identify who is holding it, it could provide personalized functionality, which could lead to a better user experience; identifying who is using a particular medical device or sensor could help provide a secure user-attestation that is useful for healthcare professionals.

VIII. Conclusion

In this paper, we introduce Zero-Effort Bilateral Recurring Authentication (ZEBRA), a novel mechanism that enables continuous authentication of users who wear a simple bracelet as an authentication token and without the need for any specialized hardware at computer terminals. Such a mechanism provides the foundation for smart deauthentication of computer terminal users in dense, dynamic work spaces. Our approach is continuous, user-agnostic, and unobtrusive. Our evaluation of ZEBRA – via a user study – shows that ZEBRA can achieve high accuracy, correctly identifying 90 % of users and locking out adversaries in less than 50 seconds. For stronger security, ZEBRA can lock out adversaries in less than 11 seconds with a small penalty in usability. ZEBRA's continuous authentication complements nearly any initial authentication (login) method and can drive a range of automatic deauthentication policies.

References

[1] R. Koppel, J. P. Metlay, A. Cohen, B. Abaluck, A. R. Localio, S. E. Kimmel, and B. L. Strom, "Role of computerized physician order entry systems in facilitating medication errors," *JAMA: The Journal of the American Medical Association*, vol. 293, no. 10, pp. 1197–1203, 2005. DOI doi:10.1001/jama.293.10.1197

[2] S. Sinclair and S. W. Smith, "Preventative directions for insider threat mitigation via access control," *Insider Attack and Cyber Security*, vol. 39, pp. 165–194, 2008. DOI 10.1007/978-0-387-77322-3_10

[3] S. W. Smith and R. Koppel, "Healthcare information technology's relativity problems: a typology of how patients' physical reality, clinicians' mental models, and healthcare information technology differ," *Journal of the American Medical Informatics Association*, no. 1-15, 2013, published Online First. DOI 10.1136/amiajnl-2012-001419

[4] S. Sinclair, "Access control in and for the real world," Ph.D. dissertation, Dartmouth College Computer Science, Hanover, NH, Nov. 2013. Available online: http://www.cs.dartmouth.edu/reports/abstracts/TR2013-745/

[5] A. Adams and M. A. Sasse, "Users are not the enemy," *Communications of the ACM*, vol. 42, no. 12, pp. 40–46, Dec. 1999. DOI 10.1145/322796.322806

[6] P. G. Inglesant and M. A. Sasse, "The true cost of unusable password policies: password use in the wild," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2010, pp. 383–392. DOI 10.1145/1753326.1753384

[7] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2012, pp. 538–552. DOI 10.1109/SP.2012.49

[8] J. Bunnell, J. Podd, R. Henderson, R. Napier, and J. Kennedy-Moffat, "Cognitive, associative and conventional passwords: Recall and guessing rates," *Computers & Security*, vol. 16, no. 7, pp. 629–641, 1997. DOI 10.1016/S0167-4048(97)00008-4

[9] J. Fernando, "The elephant in the room: Health information system security and the user-level environment," in *Proceedings of the International Conference for Internet Technology and Secured Transactions (ICITST)*, Nov. 2009, pp. 1–7. Available online: http://ieeexplore.ieee.org/xpl/abstractMetrics.jsp?arnumber=5402503

[10] Y. B. Choi, K. E. Capitan, J. S. Krause, and M. M. Streeper, "Challenges associated with privacy in healthcare industry: Implementation of HIPAA and security rules," *Journal of Medical Systems*, vol. 30, no. 1, pp. 57–64, Feb. 2006. DOI 10.1007/s10916-006-7405-0

[11] Y. Wang, S. W. Smith, and A. Gettinger, "Access control hygiene and the empathy gap in medical IT," in *Proceedings of the USENIX Workshop on Health Security (HealthSec)*, 2012, pp. 12–12. Available online: http://www.cs.dartmouth.edu/~sws/pubs/wsg12.pdf

[12] Q. Wang and H. Jin, "Usable authentication for electronic healthcare systems," in *Proceedings of the Symposium On Usable Privacy and Security (SOUPS)*, 2008, poster. Available online: http://cups.cs.cmu.edu/soups/2008/posters/wang.pdf

[13] M. D. Corner and B. D. Noble, "Zero-interaction authentication," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*, 2002, pp. 1–11. DOI 10.1145/570645.570647

[14] T. C. Meng, P. Gupta, and D. Gao, "I can be you: Questioning the use of keystroke dynamics as biometrics," in *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, 2013. Available online: http://flyer.sis.smu.edu.sg/ndss13-tey.pdf

[15] K. B. Rasmussen, M. Roeschlin, I. Martinovic, and G. Tsudik, "Authentication using pulse-response biometrics," in *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, 2014. Available online: http://www.internetsociety.org/sites/default/files/08_1_0.pdf

[16] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2012, pp. 553–567. DOI 10.1109/SP.2012.44

[17] S. Mare, A. Molina-Markham, C. Cornelius, R. Peterson, and D. Kotz, "ZEBRA: Zero-Effort Bilateral Recurring Authentication," Dartmouth College, Computer Science, Hanover, NH, Tech. Rep. TR2014-748, Mar. 2014. Available online: http://www.cs.dartmouth.edu/reports/abstracts/TR2014-748/

[18] R. Mayrhofer and H. Gellersen, "Shake well before use: Intuitive and secure pairing of mobile devices," *IEEE Transactions on Mobile Computing*, vol. 8, no. 6, pp. 792–806, Jun. 2009. DOI 10.1109/TMC.2009.51

[19] "Bump." Available online: https://bu.mp

[20] "FDA Code of Regulations, Title 21," Apr. 2013, 21CFR11.10. Available online: http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfCFR/CFRSearch.cfm?fr=11.10

[21] "FDA Code of Regulations, Title 21," Apr. 2013, 21CFR11.300. Available online: http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfCFR/CFRSearch.cfm?fr=11.300

[22] C. T. Cornelius, R. Peterson, J. Skinner, R. J. Halter, and D. F. Kotz, "A wearable system that knows who wears it," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Jun. 2014.

[23] A. Kumar, N. Saxena, G. Tsudik, and E. Uzun, "A comparative study of secure device pairing methods," *Pervasive and Mobile Computing*, vol. 5, no. 6, pp. 734–749, 2009. DOI 10.1016/j.pmcj.2009.07.008

[24] C. Cornelius and D. Kotz, "Recognizing whether sensors are on the same body," *Journal of Pervasive and Mobile Computing*, Jun. 2012. DOI 10.1016/j.pmcj.2012.06.005

[25] "8 million leaked passwords connected to LinkedIn, dating website." Available online: http://arstechnica.com/security/2012/06/8-million-leaked-passwords-connected-to-linkedin/

[26] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman, "Activity recognition from accelerometer data," in *Proceedings of the Conference on Innovative Applications of Artificial Intelligence*, vol. 3, 2005, pp. 1541–1546. Available online: http://www.aaai.org/Papers/IAAI/2005/IAAI05-013

[27] "Shimmer Research." Available online: http://www.shimmer-research.com

[28] F. Stajano, "Pico: No more passwords!" in *Security Protocols XIX*, B. Christianson, B. Crispo, J. Malcolm, and F. Stajano, Eds. Springer-Verlag Berlin, Mar. 2011, vol. 7114. DOI 10.1007/978-3-642-25867-1_6

[29] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm, "iSpy: automatic reconstruction of typed input from compromising reflections," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2011, pp. 527–536. DOI 10.1145/2046707.2046769

[30] G. Shah, A. Molina, M. Blaze *et al.*, "Keyboards and covert channels," in *Proceedings of the 15th conference on USENIX Security Symposium*, vol. 15, 2006, p. 5.