Hacksaw: Biometric-Free Non-Stop Web Authentication in an Emerging World of Wearables

Prakash Shrestha
University of Alabama at Birmingham
prakashs@uab.edu

Nitesh Saxena
University of Alabama at Birmingham
saxena@uab.edu

ABSTRACT

The currently deployed web authentication model, involving only *entry-point authentication* of users, does not do anything to protect against *account takeover attacks*. Once the attacker has compromised the entry-point authentication method, such as by learning a user's password or even two-factor authentication credentials via widely exploited mechanisms such as phishing and password database breaches, or has hijacked a login session, he can fully access and abuse the user's account and associated services. To respond to this critical vulnerability, we introduce the notion of *non-stop post-entry authentication*, to be integrated with any entry-point authentication method, using which the web service can proactively authenticate the user *throughout* the login session invisibly in the background without explicit user involvement and without the need for storing user-specific templates (like in biometric systems) thereby preserving user privacy.

We design a *transparent* and *privacy-preserving* non-stop authentication system, called *Hacksaw*, using a wrist-worn personal wearable device that authenticates the user continually by *correlating* the input events on the website (e.g., keyboard and mouse activities) with the user's corresponding hand movements captured via the device's motion sensors. Specifically, at its core, Hacksaw's correlation algorithm computes the *cosine similarity* of the *hand gesture* with the stored *generic* (i.e., non user-specific) templates of input gestures. We build an instance of Hacksaw's implementation on an Android smartwatch as the wearable and desktops/laptops as the client terminals, and comprehensively evaluate it under benign and adversarial settings. Our results suggest that Hacksaw can keep the legitimate users logged into their accounts for long durations, while promptly detecting or automatically deauthenticating remote and proximity attackers attempting to take over the users' account following the compromise of the initial login credentials or hijacking of the login session. Given that wrist-worn wearable devices are already increasingly used in many domains of daily lives (including security applications), we believe that Hacksaw can be incorporated to the current web authentication model, especially to sensitive web services such as banking or e-commerce, to significantly improve its security against online fraud, without additional effort from the users and without degrading user privacy.

WiSec '20, July 8–10, 2020, Linz (Virtual Event), Austria
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8006-5/20/07…$15.00
https://doi.org/10.1145/3395351.3399366

CCS CONCEPTS

• **Security and privacy → Authentication**.

KEYWORDS

Non-Stop Authentication, Account Takeover, Wrist-Worn Wearable

ACM Reference Format:
Prakash Shrestha and Nitesh Saxena. 2020. Hacksaw: Biometric-Free Non-Stop Web Authentication in an Emerging World of Wearables. In *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20), July 8–10, 2020, Linz (Virtual Event), Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3395351.3399366

1 INTRODUCTION

User authentication is undoubtedly one of the most crucial security components of any online service. It is supposed to allow the web service to ensure that only the legitimate, pre-registered users can gain access to their web accounts. The most commonly deployed form of web authentication uses passwords, either alone, or in combination with a second factor such as a mobile phone or a hardware token for improved security (i.e., two-factor authentication or TFA).

These almost universally deployed authentication schemes, by the nature of their design, only offer *entry-point authentication* or *one-time, initial authentication*, allowing the user to remain authenticated until the user manually logs out (i.e., '*deauthenticates*') himself, or the session gets expired (i.e., 'times out') after a sufficiently long period of inactivity. Unfortunately, many of the widely deployed entry-point authentication schemes, i.e., passwords and TFA, get compromised relatively easily on a routine basis, for instance, through the leakage of passwords via phishing attacks or hacked password databases, hacking of TFA schemes (e.g., through phishing attacks [20, 34, 44], man-in-the-browser and man-in-the-mobile attacks utilizing cross-platform services [9, 24]), or web session hijacking attacks (e.g., through session sniffing, cross-site scripting, malicious JavaScript codes, and man-in-the-middle attacks [7, 22]). Further, widely-used hardware token based TFA schemes (e.g., FIDO U2F [3]) are also vulnerable to a man-in-the-machine attack [6]. Once the initial authentication method has been compromised, the attacker (either remote, or proximity-based) can take over the user's web account, allowing the attacker complete freedom to abuse the account and the associated services.

Such a threat of unauthorized access through account takeover is highly prevalent in the online scenario. Also, the consequences of such web account takeover can be very devastating in practice. For example, the attacker with access to user's online account can take critical actions on the user's behalf (e.g., transfer funds from the bank account, make hefty purchases or send crafted emails), snoop through user's personal information, steal user's sensitive data, and modify or delete user's online data [2, 11, 12, 15, 16, 30, 31]. Given the severity of the threat of online account takeovers, it is clear that the currently deployed web authentication model involving only the entry-point authentication is insufficient. In other words, it has become paramount to adopt a *post-entry authentication mechanism* that continues to transparently recognize an already logged in user in (ideally) a *non-stop* manner, i.e., *while the user is accessing the account*, and potentially promptly deauthenticate the user or take another preventive measure if an account takeover is detected.

In this paper, we propose a *transparent* and *privacy-preserving* post-entry non-stop authentication system for the online scenario that can effectively thwart the account takeover threat. Specifically, we propose *Hacksaw*, [1], a non-stop authentication scheme based on a *wrist-wearable*, a wrist-worn device (e.g., a watch/bracelet). In Hacksaw, once the user logs into his online account using any of the standard entry-point authentication methods, the corresponding web-server continuously, yet transparently, verifies the legitimacy of the user by correlating the sequence of events observed on the online account (e.g., keyboard/mouse interactions) with the sequence of events inferred based on the wrist-motions captured by the wrist-wearable. If these two activities do not match, when a different user (attacker) accesses the online account while the victim user is performing other hand activities, Hacksaw takes proactive actions by either putting the account into a "restricted access state", alerting the user with some out of band mechanisms, or promptly deauthenticating the user. Hacksaw is designed to defeat both (targeted or untargeted) remote as well as proximity-based attacks that may attempt to take over the user's account following the compromise of initial authentication credentials, without user's knowledge.

Although Hacksaw utilizes wrist-movements captured by the wrist-wearable for post-entry non-stop authentication, it is *not a biometric scheme* because it does not rely on the user-intrinsic wrist-movements, rather, Hacksaw continuously re-authenticates the user by correlating the interactions observed on the terminal with the wrist-movements of the user. Given this, Hacksaw does not store any sensitive information about the user (unlike traditional biometrics schemes), and hence preserves the privacy of the user with respect to the leakage of authentication templates.

Hacksaw requires the user to wear a wrist-wearable equipped with motion sensors (e.g., accelerometer and gyroscope) on his mouse handling hand. The wrist-wearable is a personal device and is registered to the user's online account (or web-server). Hacksaw can work in conjunction with any of the other initial authentication schemes (e.g., the password or the TFA schemes) implemented in an online account. Although the web services can be accessed through various types of client terminals and non-stop authentication is important for all these devices, our focus in this work is on the desktop and laptop terminals. Since Hacksaw is a high-security authentication scheme, we do not envision that it would be deployed on all web services, but we expect it to be adopted on several of them especially those that are highly sensitive (such as banking or e-commerce based). The same wearable device may be used as a second factor as part of traditional entry-point TFA (e.g., using one-time PINs or active sounds [39]) as well as for Hacksaw post-entry authentication.

[1]HACKSAW denotes HACK-resistant non-Stop Authentication for the Web using wearable devices. It can be viewed as a "saw" against web account "hacks".

Our Contributions: We make the following contributions:

(1) ***A New Non-Stop Authentication Notion for the Web based on a Wearable Device:*** We introduce the idea of transparent and privacy-preserving non-stop authentication for the web based on a wrist-worn wearable device, giving rise to a concrete instantiation, the Hacksaw system.
(2) ***Design and Implementation of Hacksaw:*** We design and implement an instance of Hacksaw for an Android smartwatch and the Chrome browser, geared for desktop and laptop. Our scheme does not require any browser plugins or changes to the browser. Our concrete design is based on the correlation of the interactions observed on the website with the wrist-movements captured by a wrist-worn wearable device. At its core, Hacksaw infers a wrist-motion to an interaction by computing its *cosine similarity* with the stored interaction templates generated from a *random group of users* (no user-specific templates are needed).
(3) ***Evaluation in Benign and Adversarial Scenarios:*** We evaluate Hacksaw for authentication/deauthentication errors in both benign and adversarial settings based on the wrist motion data collected from 25 volunteer participants while they performed a web-form filling task. Our results show that Hacksaw can effectively identify the authorized users and promptly recognize the remote or proximity adversaries (within 15-20 seconds in most cases). Based on these results, we believe that Hacksaw is a viable mechanism of seamless non-stop authentication that can resist most real-world attacks.

2 SYSTEM AND ADVERSARIAL MODELS

2.1 System Model

Hacksaw considers an *online non-stop authentication system* based on a wrist-worn wearable device $\mathcal{W}$ (e.g., watch, bracelet). Hacksaw assumes that the user $\mathcal{U}$ has an online account $\mathcal{OA}$ with a remote web-service (e.g., banking, credit card, email, healthcare, utility, etc.). $\mathcal{U}$ wears $\mathcal{W}$ when accessing his $\mathcal{OA}$ through a web-browser (e.g., Mozilla Firefox, Google Chrome, etc.) on a terminal. Hacksaw assumes that there exists a separate initial authentication system (e.g., the ones based on username and password) that $\mathcal{U}$ uses to log in to $\mathcal{OA}$. Once $\mathcal{U}$ logs in to his $\mathcal{OA}$, Hacksaw continuously verifies that the current $\mathcal{U}$ is the same $\mathcal{U}$ who has initially logged in. When a different $\mathcal{U}$ attempts to use $\mathcal{OA}$, Hacksaw takes proactive actions such as putting the account into a "restricted access state", alerting $\mathcal{U}$ with some out of band mechanisms, or logging out the current $\mathcal{U}$, thereby preventing $\mathcal{OA}$ misuse. Hacksaw does so by correlating observed activities on $\mathcal{OA}$ with the activities inferred based on the wrist-motion captured by the $\mathcal{W}$ device.

In this study, we focus on the desktop PC and laptop. Specifically, we assume that $\mathcal{U}$ utilizes either a desktop PC with an external keyboard and a mouse (i.e., the *desktop setting*), or laptop with a built-in keyboard and touchpad (i.e., the *laptop setting*) to access his $\mathcal{OA}$. Instead of using the built-in touchpad, $\mathcal{U}$ may also use the external mouse with the laptop. This setup will be similar to our desktop scenario. We also assume that each $\mathcal{U}$ owns a $\mathcal{W}$ device equipped with accelerometer and gyroscope sensors and a wireless radio (e.g., Bluetooth) that it uses to communicate with its companion device, i.e., the smartphone $\mathcal{P}$. Many of today's $\mathcal{W}$ devices meets this assumption. For instance, $\mathcal{W}$ devices like fitness bands from Fitbit [21], smartwatches such as LG G Watch R [25], Sony SmartWatch [41], etc., work in conjunction with the $\mathcal{P}$ device, and all communication between them typically goes through Bluetooth. They usually come with on-board accelerometer and gyroscope sensors and can have long battery life.

We assume that $\mathcal{U}$ wears $\mathcal{W}$ on his mouse holding hand, i.e., the hand used for controlling the mouse. Further, we assume $\mathcal{W}$ as a personal device and $\mathcal{U}$ does not share it with anyone. Moreover, we assume that $\mathcal{W}$ and $\mathcal{P}$ are already registered to $\mathcal{OA}$, and they share an encryption key that they can use to secure their communication. The registration of $\mathcal{W}$ and $\mathcal{P}$ is a one-time task, and during this process, they share a secret encryption key. We also assume that all communication between devices involved in Hacksaw is secured by a secure cryptographic mechanism (e.g., HTTPS, SSL, TLS, etc.).

2.2 Adversarial Model

In an online scenario, account takeover threat, i.e., gaining unauthorized access to $\mathcal{OA}$, is quite popular. With such unauthorized access, the adversary can perform various nefarious activities, e.g., banking transaction or an online purchase, on behalf of the user. Hacksaw is designed to thwart such unauthorized access to $\mathcal{OA}$. Hacksaw considers two types of adversaries – *remote attacker* and *proximity attacker*, as described below.

(1) Remote Attacker: We consider a remote attacker who has gained unauthorized access to $\mathcal{OA}$ and attempts to misuse it remotely at a random point in time. To take over $\mathcal{OA}$, a remote attacker can hack into the initial authentication token of victim's $\mathcal{OA}$ or hijack his online web-session. Secure $\mathcal{OA}$ often implements a TFA mechanism to make their system secure. Most commonly used form of TFA requires one extra authentication token (e.g., a PIN sent/generated on the user's phone) in addition to the password to authenticate to $\mathcal{OA}$. Both of these authentication tokens can be compromised through phishing [20, 34, 44], man-in-the-browser and man-in-the-mobile attack utilizing cross-platform services [9, 24], or by other means. Besides targeting the initial authentication, a remote attacker can also steal or predict a valid web-session key to gain unauthorized access to $\mathcal{OA}$. $\mathcal{OA}$ typically maintains a session key, also known as session ID, to authenticate the user throughout the user's interaction with the account. This session token can be compromised through session sniffing, cross-site scripting, malicious JavaScript codes, and man-in-the-middle attack [7, 22]. Such a session hijacking attack is prevalent in the online setting. Further, we assume that the attacker can compromise $\mathcal{U}$'s terminal (or browser) and manipulate the input generation by tampering the JavaScript modules related to Hacksaw. However, the manipulated inputs would likely not match with the user's wrist activities, and the attack would likely fail.

Given the remote nature of the attack, the attacker remains far-off from the $\mathcal{U}$'s locality and does not have any clue what $\mathcal{U}$ might be doing. Based on the real-life activities of $\mathcal{U}$, we consider various scenarios for the remote attack. For instance, we consider a scenario where $\mathcal{U}$ is using his personal computer or his laptop at home or other places. We term this as *using-terminal* scenario. We also consider *using-phone* scenario where $\mathcal{U}$ is using his mobile phone for various purposes such as reading/writing a text, playing games, etc. Next, we consider the *writing* scenario where $\mathcal{U}$ performs a hand-writing task. In daily life, $\mathcal{U}$ also performs walking activities so we consider the attack setting with the *walking* scenario. Further, we consider *miscellaneous* wrist activities that represent an attack scenario where $\mathcal{U}$ is having a conversation with his colleagues while standing or sitting on a chair/sofa and moving his hand for random activities. During these activities, we assume that $\mathcal{U}$ is wearing $\mathcal{W}$.

(2) Proximity Attacker: We also consider a proximity attacker who has compromised $\mathcal{OA}$ similar to a remote attacker, but he tries to access $\mathcal{OA}$ when he is nearby $\mathcal{U}$. A proximity attacker can use the approach similar to the one used by the remote attacker to take over $\mathcal{OA}$. Given the fact that users often forgot to log out their accounts and lock their terminals, the proximity attacker can also wait for the targeted $\mathcal{U}$ to leave the terminal without logging out of the account. To address such a case, $\mathcal{OA}$ generally implements a time-out based approach where the logged in $\mathcal{U}$ is deauthenticated after a sufficiently long period of inactivity. During this inactivity period, anyone can have access to $\mathcal{U}$'s terminal and his $\mathcal{OA}$.

As the proximity attacker remains nearby $\mathcal{U}$, we assume that he can have audio-visual cues on $\mathcal{U}$'s activities. Further, we assume that the proximity attacker is clever and tries to access $\mathcal{OA}$ when $\mathcal{U}$ is using his desktop PC or laptop. In such a scenario, the proximity attacker can clearly see (or hear) $\mathcal{U}$ interacting with the terminal. Based on the observed interactions, the proximity attacker attempts to mimic $\mathcal{U}$'s activities to fool the Hacksaw system into treating him as a legitimate $\mathcal{U}$. Moreover, we assume that the proximity attacker employs the opportunistic strategy as proposed in [19] to break the security of Hacksaw. Specifically, based on the *audio-visual* cues, the proximity attacker tries to mimic a subset of keyboard interactions of $\mathcal{U}$, termed as *opportunistic keyboard-only attack*. If the visual access to $\mathcal{U}$'s activities is blocked, for instance, through the use of visual barrier around the terminal, the proximity attacker can still launch the opportunistic keyboard-only attack based on the audio cues alone. This is termed as *audio-only opportunistic keyboard-only attack*. These opportunistic proximity attacks represent quite a strong model as the attacker can closely observe and mimic the victim's terminal interactions. This is otherwise hard to do in real life due to the use of visual barriers, and the presence of ambient audio noises, e.g., chatter sounds from surrounding people, music, or keyboard sounds from non-targeted terminals, may mask targeted keyboard sounds. Such an opportunistic strategy was shown to be highly successful against ZEBRA [27], a local non-stop authentication system. However, Hacksaw can effectively thwart such opportunistic attackers (as demonstrated in Section 6.2.3).

3 SYSTEM ARCHITECTURE

3.1 Interactions in Hacksaw

We consider three different user-terminal interactions as follows. Appendix A Figure 8 shows the acceleration generated on the user's wrist when he interacts with the desktop.

❶ *Typing:* When a user types a character using a keyboard, the browser, HTML Document Object Model (DOM) to be specific, dispatches two events – onkeydown and onkeyup. Typing interaction is defined as the series of such onkeydown and onkeyup events.

❷ *Scrolling:* When a user rolls the wheel up or down to scroll a web-page, HTML DOM fires a series of `onwheel` events. Scrolling event is thus defined as a sequence of uninterrupted `onwheel` events.

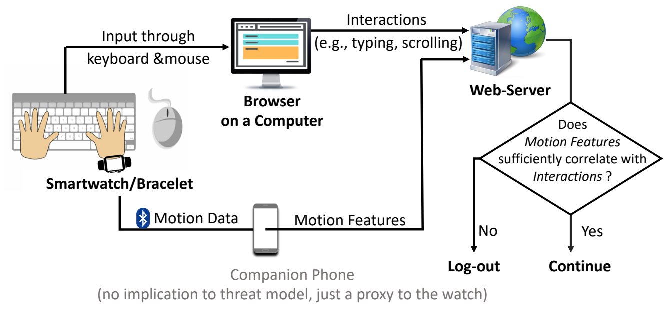


Figure 1: A high-level architecture of Hacksaw.

Unlike the desktop setting, scrolling on a laptop can be performed in various ways based on the laptop model and its configuration (e..g, using slider at the right side of the touchpad, using *TrackPoint*, or via a special (one or two) finger gesture). In our analysis, we did not find much contribution of scrolling on the performance of our Hacksaw system in the laptop setting. Therefore, for the laptop setting, we do not consider scrolling interaction.

❸ *K2M and M2K:* When a user switches from the keyboard to the mouse (or the touchpad), the user makes a hand movement (i.e., the hand used for mouse handling). This hand movement is captured by the ***K**eyboard-to-**M**ouse (K2M)* interaction. Similarly, the hand movement when the user switches from the mouse (or the touchpad) to the keyboard is captured by the ***M**ouse-to-**K**eyboard (M2K)* interaction. As K2M and M2K refer to two different hand-movements that are mirrored to each other, we combine these two events into one and refer them by K2M alone. To identify an M2K interaction using only one watch, we divide the keys on the keyboard into three regions – left region, middle region, and right region (as shown in Appendix B Figure 9). We assume that all users follow standard two-handed guidelines. We consider M2K interaction only when the user uses a key on the right region of the keyboard (i.e., the side closer to the mouse-handling hand) following the mouse event.

3.2 Real-world Architecture of Hacksaw

Figure 1 shows a high-level architectural design of our Hacksaw system. Hacksaw consists of following four main entities.

❶ *Website:* This entity is simply a browser-based application through which a user accesses his account associated with a remote web-server. After the successful entry-point authentication of the user, Hacksaw gets activated and triggers the user's wrist-wearable to transmit motion data to its companion phone. Since a website (and web-server) cannot directly communicate with the user's wrist-wearable, it first contacts the phone, which in turn connects to the wrist-wearable. The website of Hacksaw implements an *Interaction Generator* (described later in Section 4) that outputs sequence of interactions (e.g., typing, scrolling, and K2M) based on the raw input events (e.g., keyboard and mouse-related events) observed on the website. An interaction from Interaction Generator consists of an interaction ID and the timestamps when the interaction starts and ends. The generated sequence of interactions, *interact info* to be specific, is then transmitted to the companion phone through the web-server. We note that *Interaction Generator* is a JavaScript module that runs solely at the client machine to generate sequence of interactions which is transmitted to the phone for further processing. The sequence of interactions *does not* reveal what exactly the user has typed, rather it shows whether user has used the keyboard or mouse. Therefore, the transmission of interactions (not the actual keys) neither leaks any sensitive information to network attackers nor to online services.

❷ *Remote Web-server:* This entity is a remote server where a web-service such as used for banking, credit-card, email, resides in this entity. To continuously verify the legitimacy of the user, the web-server of Hacksaw implements *Decision Module*, which is responsible for making decisions on whether the current user is an authorized user or not. The web-server correlates the *interact info* from the website with the interaction inferred based on the wrist-movement captured by the wrist-wearable. If these two do not correlate sufficiently, the web-server makes the decision that its current user and the wrist-wearable user are different, and takes appropriate actions towards the current user. The web-server also forwards *interact info* from the website to the user's phone. Several cloud messaging services, e.g., Firebase Cloud Messaging (FCM) API [17], Apple Push Notification (APN) API [4], and Microsoft Push Notification Service (WNS) API [28], can be utilized to establish a communication channel between a web-server and a phone. Many security systems, such as Google 2SV [18] and Duo Push [10], are already using such infrastructures.

❸ *Phone-app:* Hacksaw consists of an application for the user's phone that processes its functionality in the background. Based on the *interact info* received from the web-server (originally from the website), phone-app extracts several characteristic features from the wrist-movements, i.e., motion sensors measurements, received from the wrist-wearable and transmits them to the web-server. We note that pre-processing the motion data in the phone and transmitting the feature vectors (not the raw motion data) to the web-server requires minimal communication bandwidth and helps preserve potential leakage of user's privacy through motion data.

❹ *Wear-app (WA):* Hacksaw consists of an application for the wrist-wearable that remains idle in the background and is activated when its wearer is authenticated to a web-service. Once activated, the wear-app starts transmitting motion readings to its companion phone. The transmission of the motion data continues until the current user logs out or gets deauthenticated from the service.

Support for Multiple Websites and Website Switching: Hacksaw can be extended to support for the scenario where the user accesses multiple instances of websites, or the user switches from one website to another. In a typical online scenario, each authenticated user session generates a unique session ID (or sequence/session number). This session ID can be tied to the Hacksaw session to track the website that requests for the re-authentication, specifically motion data from the wrist-wearable through the web-server. Given the fact that the user can interact with only one instance of the website at a given time, re-authentication would be requested from only one website that the user is currently using, and wrist-wearable (and phone) would know which web-server to respond to based on the source of the request and the session ID. Thus, a single wrist-wearable (with its companion phone) can be used for multiple instances of websites and in the website switching scenario.

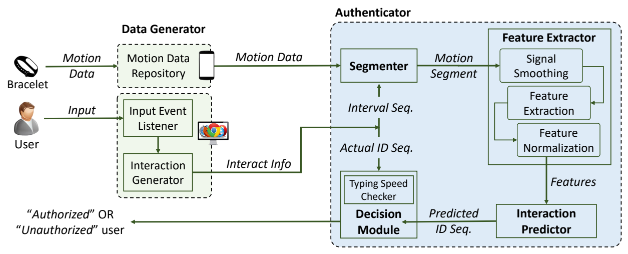


Figure 2: Prototype implementation of Hacksaw. 'Interact' indicates 'interaction' and 'Seq.' indicates 'sequence'.

Battery Consumption: Since Hacksaw requires continuous streaming of motion data from the wrist-wearable to the companion smartphone, one may suspect that it may drain the battery power of these devices. However, this is not the case because many of the apps currently available on the app stores for the phones (e.g., Google Fit, Pedometer, Step Counter, etc.) and corresponding apps on their connected wrist-wearable stream their data back-and-forth continuously without consuming much power via low power Bluetooth [26, 32]. In our prototype, wrist-wearable continuously transmits motion sensor data at the sampling rate of 200Hz, which may seem power-draining. However, as per [1] (Table V), such transmission at the sampling rate of 100Hz for 10 minutes drains only 2.1% of battery power. We expect at the sampling rate of 200Hz, the power consumption would only be a bit higher (nearly 2.5%). Moreover, power consumption can be further optimized by: (i) transmitting only feature vectors, instead of entire raw motion data, from the wrist-wearable to the phone, and (ii) limiting data transfer only to an active web-session.

4 DESIGN AND IMPLEMENTATION

As for the prototype design and implementation, and later for testing of Hacksaw, we used a widely available smartwatch LG G Watch R as a wrist-wearable, Samsung Galaxy S6 as the smartphone, a desktop PC with an external keyboard and a mouse, and Lenovo Thinkpad W530 laptop (with *bottom-centered* touchpad) as a terminal. Our implementation consists of two units – *(i) Data Generator* and *(ii) Authenticator*, as described below. Figure 2 provides the visualization of our prototype implementation.

4.1 Data Generator

I. Website and Web-Server: We developed a simple website and a web-server using HTML, JavaScript, CSS, and PHP. The website has a simple login-form and a web-form. Once the user is authenticated after providing correct username and password in the login-form, the website sends "start" push message to the Android phone which then triggers streaming the motion sensors measurements from the watch to the phone. Hacksaw uses GCM to send push message from the website to the designated phone. Then, the website leads the user to a web-form containing several generic questions about the user. The purpose of this web-form is to simulate the real-world scenario where the user accesses his online account.

The website implements *Input Event Listener* and *Interaction Generator*. Input Event Listener listens to all the input events (e.g., key presses and releases, mouse clicks, moves and scrolls, and other keyboard-mouse related events) observed on the website and feeds them to Interaction Generator. From the stream of input events, Interaction Generator creates a series of interactions. Interaction Generator considers three parameters when extracting interactions from input events – *minimum duration, maximum duration* and *idle threshold.* Minimum duration refers to the least duration that an interaction should last for. We ignore all the interaction that last for less than the minimum duration. If an interaction exceeds the maximum duration, we split it into two consecutive interactions. Both of the two newly formed interactions should meet the minimum duration constraint, otherwise, we do not split the interaction. Idle threshold is the maximum allowable time difference between two consecutive events in an interaction. If time difference exceeds the idle threshold, two consecutive events are assigned to two different interactions. The motive behind using idle threshold is to capture only the interactions that involve the user's continuous interaction with the terminal and remove the user's interactions other than using the keyboard and the mouse/touchpad. When there is a pause, and no event is observed on the terminal, it is not certain what the user might be doing. For that reason, we cannot correlate the user's activities with his wrist movement during the pause. Given this, we split the series of events into two interactions separated by the pause when the pause duration is higher than the idle threshold.

From a given series of input events, Interaction Generator outputs a sequence of interactions based on the aforementioned constraints. This sequence of interactions takes the form of

$$(id_1,\, t_{s1},\, t_{e1}),\, (id_2,\, t_{s2},\, t_{e2}),\, \ldots \tag{1}$$

where (id, t_s, t_e) represents an interaction of type id (one of the three interactions – typing, scrolling or K2M) that starts at time t_s and ends at time t_e. We term it as *"interact info"*. The time duration from t_s to t_e constitutes the *interaction interval* for the interaction 'id'. All this sequence of interactions is uploaded to the web-server for the purpose of offline analysis in our current implementation.

II. Phone and Watch Apps: We built two Android apps, one for the phone and another for the watch, that stay idle in the background. A "start" GCM signal activates the phone app, which in turn activates the watch app over the Bluetooth channel. Similarly, a "stop" GCM signal stops both apps. Once activated, the watch app starts streaming accelerometer and gyroscope sensors readings to the phone at 200Hz, which is the standard sampling rate of the current generation of wearable devices. The accelerometer data from the watch is transmitted and stored in the form of

$$(t_i,\, x_i,\, y_i,\, z_i),\, (t_{i+1},\, x_{i+2},\, y_{i+2},\, z_{i+1}),\, \ldots \tag{2}$$

where (t_i, x_i, y_i, z_i) represents one accelerometer data sample captured at time t_i and x_i, y_i, z_i are instantaneous acceleration along x-, y- and z-axes, respectively. The gyroscope data also takes the similar form. In our current implementation, these sensors readings are stored locally in the phone for further analysis.

4.2 Authenticator

As shown in Figure 2, Authenticator consists of four main components – *Segmenter, Feature Extractor, Interaction Predictor*, and *Decision Module.* We implemented all these components in Matlab. In the real-world implementation, the phone would implement Segmenter and Feature Extractor, and the web-server would implement Interaction Predictor and Decision Module.

I. Segmenter: This component takes two inputs – (i) the accelerometer and gyroscope readings from the watch, and (ii) the sequence of interaction intervals from Interaction Generator. For an interaction interval (t_s, t_e), Segmenter selects all the accelerometer and gyroscope data samples with time $t : t_s \le t \le t_e$ and forms a block. Thus, Segmenter creates a series of blocks for a given sequence of interaction intervals and are sent to *Feature Extractor.* Motion data samples that are outside the interaction intervals are discarded.

II. Feature Extractor: This component receives a series of blocks of motion data from Segmenter and computes 150 statistical features over each block of motion data (details are provided in Appendix C). Each of the extracted features is re-scaled in the range of (0, 1) using standard min-max normalization. Feature Extractor thus forms a feature vector $F = (f_1, f_2, ..., f_{149}, f_{150})$, and sends the sequence of feature vectors $F_i, F_{i+1}, F_{i+2}, ...$, each corresponding to one interaction block, to Interaction Predictor.

III. Interaction Predictor: Based on the sequence of feature vectors received from Feature Extractor, Interaction Predictor infers corresponding interactions, resulting in a *predicted interaction ID sequence.* Interaction Predictor provides its prediction based on the similarity of interaction's feature vector with stored generic templates of interactions. Below we describe how interaction templates are generated and provide details on the working algorithm of Interaction Predictor.

ⓐ Interaction Template Generation: An interaction template of a user consists of three types of signatures, each corresponding to one of the three interactions, i.e., typing, scrolling, and K2M. A signature of an interaction is a set of characteristic features (or feature vector) corresponding to that interaction that we computed as follow. Feature vectors corresponding to each of the interactions are grouped together, and centroid (here, we use average) of each feature in feature vectors is computed. The resultant centroids of features for each interaction form the signature of that interaction. Since we consider only typing and K2M interaction for the laptop setting, an interaction template for the laptop contains only two types of signature corresponding to these two interactions. Interaction Predictor employs *n* interaction templates created from a random group of *n* users to predict a motion data block. It does not require user intrinsic signature for interaction, instead, it uses generic interaction signatures from a random group of users for prediction. In a real-world implementation, a web service can use (or hire) a small representative set of random users to build such interaction templates, or re-use the templates built by other services with a similar pool of users.

ⓑ Working Algorithm of Predictor: Based on the feature vector received from Feature Extractor, Interaction Predictor outputs an interaction ID. Predictor does so based on the majority voting from pre-built templates (as described earlier) consisting of interaction templates from '*n*' users. For each user template, Predictor computes the *cosine similarity* of the supplied feature vector to each of the three interaction signatures of the template. Since the interaction similarity in Hacksaw relies on the features' orientation rather than its magnitude (given normalized feature vectors), cosine similarity is a good choice in such a scenario to compute a similarity measure. Predictor outputs interaction $i : i \epsilon \{typing, scrolling, K2M\}$ for a given feature vector if its similarity score with the signature of interaction i of the template is higher than that with the rest of the interaction signatures. If a feature vector corresponds to an interaction other than typing, K2M, and scrolling, then its similarity score to each of the stored templates would be extremely low. Therefore, we employ *interaction similarity threshold* ($\theta_{interact}$) to identify such '*other*' interactions. If the similarity score is below $\theta_{interact}$, predictor outputs *other.* '*n*' templates thus generate *n* different predictions for a supplied feature vector, and final prediction is computed based on the majority voting on the predictions.

IV. Decision Module: Decision Module receives the sequence of actual interactions, specifically the *actual interaction ID sequence*, from the Interaction Generator and the sequence of interactions inferred by the Interaction Predictor based on the user's wrist movement, i.e., *predicted interaction ID sequence.* Decision Module correlates these two sequences, and outputs '1' if the two sequences match, otherwise, outputs '0'. When correlating two sequences, Decision Module compares '*w*' interactions at a time. We term it as *window size.* As a measure of how well two sequences match, Decision Module computes a similarity score s_{win} ($0 \le s_{win} \le 1$) for each window as follow

$$s_{win} = \frac{no.\ of\ matching\ interactions}{window\ size\ (w)} \tag{3}$$

where '$no.\ of\ matching\ interactions$' represents the number of predicted interaction ID that matches with the actual interaction ID in a window. To make a final verdict for a window, Decision Module checks the similarity score s_{win} against similarity threshold (θ_{win}). If s_{win} is greater than θ_{win}, decision module concludes the current terminal user and the wrist-wearable user are the same for that window. Otherwise, it concludes that the two users are different.

Decision Module may incorrectly output '0' for a legitimate window. In such a case, if Hacksaw immediately deauthenticates the user, it would hamper user experience. To minimize such false negatives, Decision Module allows '*b*' consecutive windows with the score of '0' before it deauthenticates the user. We term it as a bonus window. If *b* is set to 2, Decision Module deauthenticates the user when it receives two consecutive windows with the score of 0. If Decision Module receives a window with the score of 1, it resets the counter of the incorrect window to 0.

Decision Module also incorporates *Typing Speed Checker* that constantly monitors the typing speed of the user. The reason behind employing Typing Speed Checker is to thwart opportunistic attackers. In our analysis, we observed that the typing speed of an opportunistic attacker is extremely lower than that of a regular user, potentially because he has to correctly mimic a subset of victim user's activities in real time. Given this, we implement Typing Speed Checker in our Decision Module. For every *n* seconds, Decision Module checks the typing speed of the user. If it finds that the typing speed of the user is below *typing threshold* (v), it raises a flag indicating the potential for an opportunistic attack. Similar to bonus window, Decision Module uses a bonus flag 'f' for typing speed. Bonus flag of f indicates that the Decision Module allows 'f' flags before it deauthenticates the user. We note that if no activity is observed on the account (i.e., for the typing speed of 0), the Decision Module does not deauthenticate the user. The user is deauthenticated only when the Decision Module observes some events on the account and the typing speed of the user is lower than the set threshold. We note that Typing Speed Checker is an independent generic component incorporated in Hacksaw to thwart the opportunistic attacks. It may or may not be added depending on whether proximity attacks of [19] are considered as a practical threat.

Thus, Decision Module uses window size (w), its similarity threshold (θ_{win}), bonus window (b), typing threshold (v), and bonus flag (f) when deciding the legitimacy for a user.

5 DATA COLLECTION

For our data collection, we recruited 25 participants (age: 20-35 years; 19 males, 6 females). Participants in our study were mostly graduate students at our university. Our pool of users consists of both dexterous typists and less-experienced ones. All of them were right-handed except one participant who was able to use both hands equally, i.e., ambidextrous. All participants own a laptop and majority of their laptops had touchpad at the bottom-center location, and few had it on the right side of the keyboard. Some of the participants use an external mouse when they use their laptops while other use a built-in touchpad. Similar number of participants and demographics are well-established in lab-based studies in biometrics research [8, 19, 27, 36, 45].

Prior to the experiment, participants were told that the purpose of our study was to collect information on how they use their wrist when interacting with a desktop or a laptop. We intentionally did not disclose the actual nature of the study because it may impact the user's natural behavior towards the system. They were informed that the wrist-motion (i.e., accelerometer and gyroscope) data, and all inputs through the keyboard and the mouse (or touchpad) events will be recorded. All these participants serve as users of the system. One member of the researcher team played the role of a strategic proximity attacker to evaluate Hacksaw against proximity attacks. The attacker was well-trained and representative of an expert in mimicking the victim's terminal activities based on audio or audio-visual cues. The experiment and data collection was approved by the IRB at our institutions. At the end of the study, we explained the actual purpose of the experiment.

Experiments were conducted in a quiet environment that provides a strong advantage to an audio-based proximity adversary. During the experiment session, each participant performed two 10-minute tasks of filling a web-form. In one task, a terminal with an external keyboard and a mouse was used that represents the desktop setting of Hacksaw, while in the other task, a laptop with built-in keyboard and touchpad was used to simulate the laptop setting. The form-filling task involves all types of interactions that occur during a web-surfing session. Although the frequency of different interactions varies based on the types of websites (e.g., email vs. online shopping site), as long as the interactions observed on the terminal match with the predicted interactions based on the wrist-activities, Hacksaw would work well.

During these tasks, the proximity attacker executed each of the two strategic attacks. In the desktop scenario, the adversary performed audio-only opportunistic attacks. In this scenario, the attacker and the user were positioned approximately 1 meter apart facing in opposite direction so that the adversary cannot see the victim but can hear the user interacting with the terminal. Based on the audio cues received when the user interacts with the desktop, the adversary attempts to mimic the victim's activities to fill out the similar form at his end but using only the keyboard. In the laptop scenario, the adversary was allowed to position himself in a way such that he has a clear visual access of the victim. Similar to the desktop setup, the adversary tries to mimic the victim's keyboard activities to fill out the form at his end based on both audio and visual cues. Thus, in the desktop scenario, the assigned adversary executed an *audio-based* keyboard-only opportunistic attack, and *audio-video based* keyboard-only opportunistic attack was executed in the laptop scenario. Thus, the 25 user sessions resulted in a total of 50 data samples, with each sample consisting of three traces: *motion data* from the user's watch, *input events* (keyboard-mouse related events information) recorded on the victim terminal, and *input events* recorded on the attacker terminal. All traces within a sample were synchronized. We note that the differences in hardware devices do not have any impact on the adversary's ability to mimic the victim's terminal activities because opportunistic attacks are executed using only the keyboard. Further, these two attacks were performed according to their description found in [19].

We also collected motion data samples for each of the following regular activities – walking, writing, using-phone, using-terminal, and miscellaneous (detailed earlier in Section 2.2) from randomly selected two participants. The purpose of this data collection is to evaluate Hacksaw against a remote attack when the victim may be performing different activities while the attacker tries to access the victim's online account. We asked two of the participants to walk in their regular walking style. Similarly, we asked another two participants to use their mobile phones in their usual style. We did not restrict them from performing any activities on the phone. They were allowed to do any regular tasks on their phones such as surfing the Internet, reading/writing text/email, making notes, setting the alarm, etc. To two of the participants, we provided a wiki link on a random topic and asked them to write its content on a sheet of paper. We also collected motion data when the user and one of the researchers were having a normal conversion on a random topic. We collected motion data in such a scenario from four user sessions. In the first two sessions, the users were sitting on a chair while in another two sessions, the users were standing. During these four user sessions, users were moving their hands in their habitual pattern. The users performed each of these activities for 5 minutes. While performing these activities, we asked the users to wear the watch to collect the motion sensor readings corresponding to these activities. Thus, we collected two data samples for each of the following regular activities – walking, writing, using-phone, using-terminal, and four data samples for miscellaneous activity.

6 EVALUATION AND RESULTS

6.1 Optimal Value Selection

To find an optimal value for different parameters of Hacksaw, we employ *False Negative Rate (FNR)*, *False Positive Rate (FPR)* and *True Negative Rate (TNR)*. FNR is the rate of incorrectly predicting positive instances as negative, FPR is the rate of incorrectly predicting negative instances as positive, and TNR is the rate of correctly identifying negative instances. We also compute *Equal Error Rate (EER)*, an equilibrium point of FNR and FPR. We note that these metrics are not the real metrics for evaluating the performance or security of our approach, rather, they help to choose optimal values for the parameters. The actual metrics to evaluate the performance of Hacksaw are described later in this section. In all our analysis, we employ *leave-one-user-out* cross-validation approach. In particular, for a given user, we build interaction predictor using samples from all other users. It indicates that the system does not require any user-specific prediction model, thereby making the model user-agnostic.

6.1.1 Interaction Similarity Threshold ($\theta_{interact}$). To find an optimal value for $\theta_{interact}$, we compute FNR and FPR for different values of similarity score. For Interaction Predictor, FNR indicates the fraction of input interactions (i.e., typing, scrolling, and K2M) that has been incorrectly predicted as different interactions. Similarly, FPR of Interaction Predictor indicates the fraction of *'other'* interactions that has been incorrectly predicted as one of the input interactions. To compute FNR, we paired each of the input interaction samples from the users when they were filling out the web-form with one of the motion data samples captured when the users were performing *other* activities. This cross-pairing process simulates the scenario where an attacker attempts to access the victim's online account when the victim is doing *'other'* activities.

As expected, we found that the performance of Interaction Predictor in the desktop setting is relatively better compared to that in the case of the laptop, i.e., the lower EER value at the higher similarity score (FPR and FNR plots for Interaction Predictor is shown in Appendix Figure 10). This is because the wrist-movement when the user uses a laptop is relatively subtle compared to the wrist-movement when the user uses a desktop. Since we do not want to log out any legitimate user when he is using a terminal, we want to have minimum FNR for interaction prediction. Therefore, we choose the similarity score where we achieve minimum FNR with reasonable FPR. We chose 0.90 as an optimal value for $\theta_{interact}$ for the desktop setting, where FNR and FPR are 0.06 and 0.11, respectively. Similarly, for the laptop setting, we chose 0.85 as an optimal value for $\theta_{interact}$, where FNR and FPR are both 0.12.

6.1.2 Window Size (w) and Similarity Threshold (θ_{win}). To find an optimal value for w and θ_{win}, we employ FNR and TNR. For Decision Module, FNR is the fraction of all windows from an authorized user that is predicted as from the unauthorized user. Similarly, TNR is the fraction of all windows from an unauthorized user that is correctly detected as from unauthorized users. To compute FNR and TNR, we follow the similar approach as earlier.

To find an optimal value for w and θ_{win}, we use the optimal value of $\theta_{interact}$ that we chose earlier, i.e., 0.90 for the desktop and 0.85 for the laptop settings. With this parameter setting, we evaluate the performance of Decision Module for different window sizes of (5-30) and different window similarity thresholds of 50-75%. Figure 3 shows the average FNR for various window sizes and window similarity thresholds for the desktop and the laptop setting. As can be seen from the figure, the average FNR decreases on increasing the window size and decreasing the similarity threshold. This indicates that more the interactions provided on a window and the lower the similarity threshold, better the Hacksaw system performs. However, at the same time, it would take a longer time to detect the adversary and the overall attack detection accuracy become lower. Therefore, it is necessary to choose an optimal window size and window similarity threshold so that Decision Module performs well in benign and adversarial settings.

As shown in Figure 3a, we achieved average FNR of less than 0.04 for the window sizes above 10 and nearly 0 for window similarity threshold of less than 65% for the desktop setting. For the laptop setting, we achieved a relatively higher average FNR, as shown in Figure 3b. The higher average FNR in the laptop setup versus the desktop setting is expected because the wrist-movement corresponding to different interactions when using the laptop is more subtle compared to the interactions when using the desktop. As can be seen from the Figure 3b, Hacksaw performs better in terms of correctly detecting authorized users for window sizes greater than 14 and similarity thresholds of less than 65%.

Similarly, Figure 4 shows the average TNR for various window sizes and window similarity thresholds for the desktop setting. We achieved similar results with the laptop settings. Hacksaw performs well in detecting adversaries as indicated by high TNR of > 0.92.

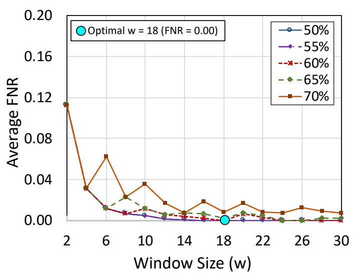
With these results, we chose window size (w) of 18 and window similarity threshold (θ_{win}) of 60% as optimal values, where FNR and TNR are 0.00 and 0.96, resp., in the desktop setting, and in the laptop setting, they are 0.01 and 0.96, resp.

6.1.3 Other Parameters. To find optimal values for the parameters other than those mentioned above, we employ the approach described in Appendix E. All the parameters and their values used in Hacksaw are presented in Table 2.

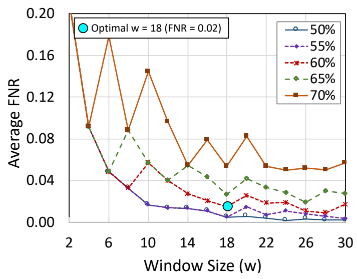
Optimal Values for Other Desktops and Laptops: Since all standard keyboards and mice in a desktop follow more or less similar configuration, we believe that the generated interaction templates and the chosen optimal values may work for all desktop scenarios. These interaction templates and optimal values would also work for all laptops with *bottom-centered* touchpad (the most common setting in laptop). In the case of the laptop with a different touchpad placement, it may require to build different interaction templates and tune the values of Hacksaw's parameters accordingly for its best performance. This is because the wrist gesture with such a different configuration would be different. However, since the interaction template generation and tuning of the parameters can be done offline, it can be performed easily, for example, by hiring a random group of users, once for each touchpad configuration.

6.2 Performance with Selected Parameters

Using the chosen optimal values for the parameters (as listed in Table 2), we evaluate the performance of Hacksaw in both benign and adversarial settings. Specifically, we use *user log out time* (*UT*) as a metric to evaluate how often Hacksaw will log out an authorized user, and *attacker log out time* (*AT*) as a metric to evaluate how quickly Hacksaw detects an unauthorized user. A large value of *UT* is desirable because it allows the authorized user to remain logged in for an extended period of time without getting accidentally logged out, and hence improves the usability. On the other hand, it is better to have a smaller value of *AT* because it quickly locks the adversary giving only a shorter window for an adversary. We measure both *UT* and *AT* in terms of the number of windows. Each window in Hacksaw consists of 18 interactions (i.e., the window size), each of 500 ms long (and max of 3500 ms for K2M). A window mostly contains typing interactions with one K2M and few scrolling events. This makes a window of length approximately 10 - 15 seconds.



(a) Desktop



(b) Laptop

Figure 3: Average FNR vs. window size (w) for different similarity threshold (θ_{win}) values. Fraction of windows that are incorrectrly classified as mismatching.

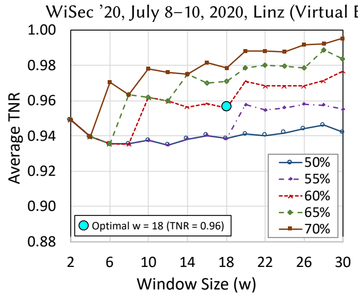
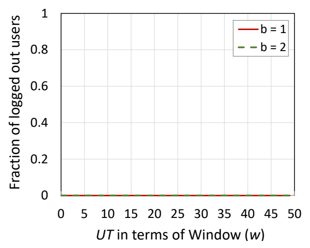
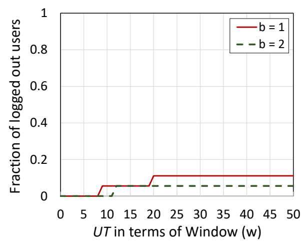


Figure 4: Desktop. Average TNR vs. window size (w) for different similarity threshold (θ_{win}) values. Fraction of windows that are correctly classified as 'other' activities. Similar results were achieved with laptop.

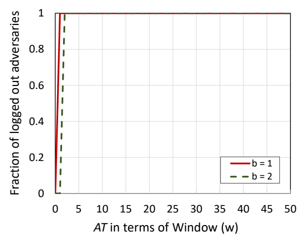


(a) Desktop (when $\theta_{interact}$ = 0.90).

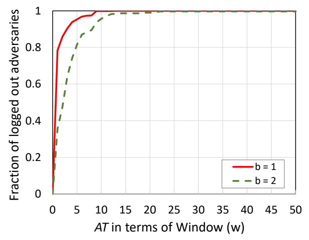


(b) Laptop (when $\theta_{interact}$ = 0.85)

Figure 5: Fraction of logged out users in a given authentication window (with w = 18, θ_{win} = 60%). 'UT': 'user log out time'.

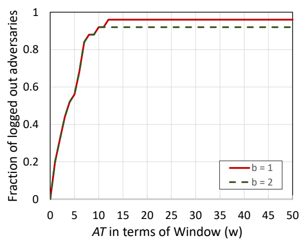


(a) Walking, Using Phone, Writing, and Miscellaneous

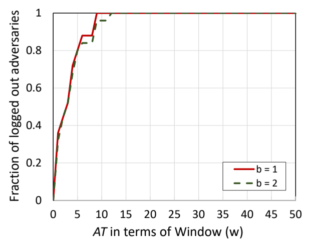


(b) Using Terminal

Figure 6: Desktop. Fraction of remote attackers logged out for a given authentication window when the victim user performs various activities. 'AT': 'attacker log out time'. Similar results were achieved with laptop.



(a) Desktop. *Audio-only* attack.



(b) Laptop. *Audio-video* attack.

Figure 7: Performance of opportunistic proximity attackers.

6.2.1 Performance with Authorized Users. Figure 5 shows the performance of Hacksaw with authorized users in the desktop and laptop settings. As observed from the Figure 5a, all the users were correctly identified as authorized users and were able to remain logged in for the entire duration of the experiment when they were using the desktop. When using the laptop, 88% (i.e., 22/25) of the users were able to remain logged in for the entire duration of the experiment when b = 1 (as shown in Figure 5b). When b = 2, 96% (i.e., 24/25) of the users were identified as authorized users, and only one user was logged out after the 12th authentication window. This was potentially because the logged out user's typing style was different from the standard two-handed typing style. These results show that our Hacksaw system can effectively identify a benign user in both the desktop and the laptop settings.

6.2.2 Resistance against Remote Attackers. Figure 6 shows the performance of Hacksaw against remote attackers when they use the desktop to launch the attack in various scenarios (as described earlier in Section 2.2). As the figure shows, the remote attackers were quickly detected and logged out within a few authentication windows. For the scenario with *walking, using-phone, writing,* and *miscellaneous* activities, all the attackers were deauthenticated within the first and second authentication window (i.e.,<25 seconds) when b was set to 1 and 2, respectively. In the *using-terminal* scenario, Hacksaw took relatively longer time to log out all the remote attackers compared to other scenarios. This is perhaps because wrist activities of the victim accidently match with those of the attacker as they both were performing the *using-terminal* activity. Fortunately, more than 90% of remote attackers were logged out within 7th authentication window (i.e., <85 seconds) when b = 2, and when b = 1, all attackers were logged out. We achieved similar results when the remote attacker uses the laptop. We note that the attack here is a remote and non-targeted, so it would be hard for the remote attacker to launch the attack at the time when the victim performs a *using-terminal* activity.

6.2.3 Resistance against Proximity Attackers. The performance of Hacksaw against strategic proximity attackers in the desktop and the laptop settings is presented in Figure 7. As can be read from the figure, Hacksaw can effectively identify such proximity attackers. In both the desktop and laptop settings, more than 90% of the attackers were kicked out in less than 10 authentication window for both b = 1 and b = 2. In the desktop setting, only one attacker (4%) was able to remain logged in for the entire duration of the experiment. We recall that such a opportunistic proximity attack represents quite a strong model where the attacker can closely observe and mimic the victim's terminal interactions, which is hard to execute in real life. In the laptop setting, all attackers were kicked out after 10 authentication windows.

6.2.4 Summary of Results. Our results show that Hacksaw can effectively identify the authorized user in both the desktop (with 0%

of lock out) and the laptop (with less than 5% of lock out) settings. When an authorized user is occasionally logged out, the user can be allowed to retry logging in with the entry-point authentication, and Hacksaw would continue to re-authenticate the user. Hacksaw can also effectively detect the remote adversary (within 15-20 seconds, in most of the cases) in both desktop and laptop settings in all the scenarios. For security sensitive online transactions with Hacksaw (e.g., a huge online purchase, or banking transaction), the web interface can be designed in such a way that forces the user to interact with the website for a sufficiently long period of time, and if the interaction while performing an activity on the website is too short (<20 seconds), the activity can be blocked. This will enable Hacksaw to even defeat advanced attackers who try to perform the activity very quickly. In sum, Hacksaw works well in both the benign and adversarial scenarios in both the desktop and laptop setting. We note that the handedness of the users will not affect the performance of Hacksaw as long as the wrist-wearable is worn on mouse-holding hand. However, the physical ailments that involves wrist movements, such as tremor due to multiple sclerosis, may affect its performance. Fortunately, we can still tune the parameters to support such situations. Accessibility has always been a challenge in many authentication systems, especially in behavioral biometrics.

7 RELATED WORK

In an attempt to mitigate account takeover threats, fraudulent login detection mechanisms have been proposed in [13, 40]. However, these schemes are based on the network and system information, e.g., IP, geo-location, and browser configuration, which can be easily impersonated by the attacker. Further, these schemes are inherently still entry-point mechanisms and therefore can not protect against account takeover through session hijacking. Behavioral biometrics techniques such as keystroke dynamics [5, 42] and mouse dynamics [14, 29, 33] have been suggested for continuous authentication in the online scenario. However, they are vulnerable to theft, internal observation attacks [43] and synthetic attacks [35], in addition require storing users' privacy-sensitive biometrics templates online.

Work related to ours is ZEBRA [27], a zero-effort bilateral deauthentication method. ZEBRA is intended for scenarios where users authenticate to a common local computer terminal (such as a desktop computer in a shared setting). Similar to Hacksaw, ZEBRA requires the user to wear a bracelet equipped with motion sensors on his mouse-holding hand. The bracelet is wirelessly connected and pre-paired 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's motion sensors. The logged-in user is deauthenticated when the two sequences do not match.

Hacksaw is different from ZEBRA in several important aspects. *First*, ZEBRA is intended for user authentication to a local machine, while Hacksaw is designed for web authentication. Being a local authentication system, the threat model of ZEBRA considers only a naive proximity attack while Hacksaw threat model considers the proximity attack as well as strong remote attacks. Specifically, Hacksaw considers a account takeover threat that covers various devastating real-world attacks including session hijacking and man-in-the-middle attacks. *Second*, Hacksaw is carefully designed to work well with the laptop setting (Section 6) while ZEBRA is designed and evaluated for only the desktop scenario. *Third*, Hacksaw uses off-the-shelf smartwatches that have a motion sensor data sampling frequency of 200Hz while ZEBRA employs the high-end Shimmer Research bracelet [37] with a sampling frequency of 500Hz. Although Hacksaw uses lower sampling frequency, it still performs better in both the benign and adversarial settings. Hacksaw correctly identified 100% authorized users (Figure 5a), while ZEBRA only identified 85% users correctly (Figure 8 in [27]) when b (or g) = 1. ZEBRA is vulnerable to proximity attacks (success rate>40%) [19], while Hacksaw can effectively (>90%) identify such attacks (Figure 7). *Fourth*, unlike ZEBRA, interaction matching algorithm of Hacksaw considers the "other" interaction representing activities other than input interactions (i.e., typing, scrolling, and K2M) that enables Hacksaw to effectively detect if the original user is indeed using the system or performing other activities.

8 CONCLUSION AND FUTURE WORK

We presented Hacksaw, a transparent and privacy-preserving non-stop web authentication system based on a wrist-worn wearable that can effectively detect and prevent the account takeover attacks. Once the user logs into a Hacksaw implemented online account, Hacksaw continually re-authenticates the user by comparing the events observed on the website with the wrist-movements of the user captured by the wrist-wearable. Our results indicate that Hacksaw works well at detecting the user and the remote and proximity attackers attempting to take over the user's online account by compromising initial login credentials or hijacking of the login session. Since Hacksaw does not require storing any sensitive information about the user (unlike traditional biometric schemes), it preserves the privacy of the user against template-hijacking attacks. Given that wrist-wearables are already gaining momentum in the user space and have already been used in security applications, Hacksaw can be incorporated to the current web authentication model without imposing any extra effort from the users throughout the login session. Hacksaw can be integrated with any initial login method, including passwords or two-factor authentication protocols, all of which are highly susceptible to account takeovers in the absence of non-stop authentication.

Since different orientations of the laptop (e.g., on the user's lap, while seated on a sofa) may generate different interaction gestures, it may impact the performance of Hacksaw. Future research is needed to explore the impact of such orientations on the performance of Hacksaw. Hacksaw may be extended to support the non-stop authentication on the personal devices other than the desktop and the laptop, e.g., mobile phones and tablets, by tweaking its algorithmic design. Since interaction and interaction gestures when using a mobile phone (or a tablet) are different from those when using a desktop (or a laptop), rigorous future research would be needed to explore more in this direction. Our current implementation is merely a prototype of the system, and is evaluated with just 25 graduate students. Much more work is needed to take it to the production level, such as building an end-to-end system, studying the feasibility of re-using the interaction templates, and evaluating the system performance with much larger and diverse pool of users.

ACKNOWLEDGMENT

The authors would like to thank our shepherd Dr. Adwait Nadkarni and anonymous reviewers for their feedback on the paper. We would also like to thank Dr. Tzipora Halevi for her valuable suggestions on a previous draft of the paper. This work is partially supported by National Science Foundation (NSF) under the grants: CNS-1547350, CNS-1526524 and CNS-1714807.

REFERENCES

[1] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Kemal Akkaya. 2018. WACA: Wearable-Assisted Continuous Authentication. *arXiv preprint arXiv:1802.10417.*

[2] Agari. 2018. Account Takeover-Based Email Attacks Increased by 126% in 2018. https://goo.gl/8C59yP.

[3] FIDO Alliance. 2017. Universal 2nd Factor (U2F) Overview. https://bit.ly/2VbXVGy.

[4] Apple Inc. 2018. APNs Overview. https://goo.gl/k37dLV. Accessed: February 1, 2018.

[5] Salil P Banerjee and Damon L Woodard. 2012. Biometric authentication and identification using keystroke dynamics: A survey. *Journal of Pattern Recognition Research* 7, 1 (2012), 116–139.

[6] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, Viswanathan Manihatty Bojan, and Tuomas Aura. 2018. Man-in-the-machine: exploiting ill-secured communication inside the computer. In *USENIX Security Symposium.* 1511–1525.

[7] Stefano Calzavara, Riccardo Focardi, Marco Squarcina, and Mauro Tempesta. 2017. Surviving the Web: A Journey into Web Session Security. *ACM Computing Surveys (CSUR)* 50, 1, 13.

[8] Mauro Conti, Irina Zachia-Zlatea, and Bruno Crispo. 2011. Mind how you answer me!: transparently authenticating the user of a smartphone when answering or placing a call. In *Symposium on Information, Computer and Communications Security.* ACM, 249–259.

[9] Alexandra Dmitrienko, Christopher Liebchen, Christian Rossow, and Ahmad-Reza Sadeghi. 2014. On the (in) security of mobile two-factor authentication. In *International Conference on Financial Cryptography and Data Security.* Springer.

[10] Duo Security Inc. 2018. Duo Mobile: Duo Security. https://duo.com/solutions/features/user-experience/easy-authentication.

[11] Forbes. 2016. Hackers Are Hijacking Phone Numbers And Breaking Into Email, Bank Accounts: How To Protect Yourself. https://goo.gl/dyV1bR.

[12] Fox News. 2018. Hackers are going after your online bank account, report says. https://goo.gl/A9TbRd.

[13] David Freeman, Sakshi Jain, Markus Dürmuth, Battista Biggio, and Giorgio Giacinto. 2016. Who Are You? A Statistical Approach to Measuring User Authenticity.. In *NDSS.* 1–15.

[14] Lex Fridman, Ariel Stolerman, Sayandeep Acharya, Patrick Brennan, Patrick Juola, Rachel Greenstadt, and Moshe Kam. 2015. Multi-modal decision fusion for continuous authentication. *Computers & Electrical Engineering* 41 (2015).

[15] Gadget 360. 2014. Hacked Email Accounts Spread Spam Faster: Study. https://goo.gl/UnNd6U.

[16] Gadget 360. 2014. Home Depot Says About 53 Million Email Addresses Stolen in Breach. https://goo.gl/DDh7Hi.

[17] Google Inc. 2018. Firebase Cloud Messaging | Firebase. https://firebase.google.com/docs/cloud-messaging/. Accessed: February 1, 2018.

[18] Google Inc. 2018. Google 2-Step Verification. https://www.google.com/landing/2step/.

[19] O Huhta, P Shrestha, S Udar, M Juuti, N Saxena, and N Asokan. 2016. Pitfalls in Designing Zero-Effort Deauthentication: Opportunistic Human Observation Attacks. In *Network and Distributed System Security Symposium (NDSS).*

[20] Matthew Humphris. 2018. Hacker Proves Bypassing Two-Factor Authentication is Easy. https://goo.gl/MbCj3W. Accessed on July 23, 2018.

[21] Fitbit Inc. 2018. Fitbit Official Site for Activity Trackers. https://www.fitbit.com/. Accessed: November 16, 2018.

[22] Vineeta Jain, Divya Rishi Sahu, and Deepak Singh Tomar. 2015. Session Hijacking: Threat Analysis and Countermeasures. In *Int. Conf. on Futuristic Trends in Computational Analysis and Knowledge Management.*

[23] Nikolaos Karapanos, Claudio Marforio, Claudio Soriente, and Srdjan Capkun. 2015. Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound.. In *USENIX Security.* 483–498.

[24] Radhesh Krishnan Konoth, Victor van der Veen, and Herbert Bos. 2016. How anywhere computing just killed your phone-based two-factor authentication. In *Conference on Financial Cryptography and Data Security.* Springer.

[25] LG Electronics. 2018. LG G Watch R. http://www.lg.com/us/smart-watches/lg-W110-lg-watch-r. Accessed: November 16, 2018.

[26] Xing Liu, Tianyu Chen, Feng Qian, Zhixiu Guo, Felix Xiaozhu Lin, Xiaofeng Wang, and Kai Chen. 2017. Characterizing smartwatch usage in the wild. In *International Conference on Mobile Systems, Applications, and Services.* ACM.

[27] Shrirang Mare, Andrés Molina Markham, Cory Cornelius, Ronald Peterson, and David Kotz. 2014. Zebra: Zero-effort bilateral recurring authentication. In *Symposium on Security and Privacy (SP.* IEEE.

[28] Microsoft Inc. 2018. Windows Push Notification Services (WNS) Overview. https://goo.gl/sTmGPK. Accessed: July 26, 2018.

[29] Soumik Mondal and Patrick Bours. 2016. Combining keystroke and mouse dynamics for continuous user authentication and identification. In *Identity, Security and Behavior Analysis (ISBA).* IEEE, 1–8.

[30] NBC News. 2018. Hundreds of Millions of Email Accounts Hacked and Traded Online, Says Expert. https://goo.gl/su6uS9.

[31] NDTV. 2014. Colombian President's email account hacked: report. https://goo.gl/3isSUu.

[32] Emirhan Poyraz and Gokhan Memik. 2016. Analyzing power consumption and characterizing user activities on smartwatches: summary. In *International Symposium on Workload Characterization (IISWC), 2016.* IEEE.

[33] Maja Pusara and Carla E Brodley. 2004. User re-authentication via mouse movements. In *Workshop on Visualization and data mining for computer security.* ACM.

[34] Gurubaran S. 2018. Hackers can Bypass Two-Factor Authentication with Phishing Attack. https://gbhackers.com/bypass-two-factor-authentication/. Accessed on July 23, 2018.

[35] Abdul Serwadda and Vir V Phoha. 2013. Examining a large keystroke biometrics dataset for statistical-attack openings. *ACM Transactions on Information and System Security (TISSEC)* 16, 2 (2013), 8.

[36] Muhammad Shahzad, Alex X Liu, and Arjmand Samuel. 2013. Secure unlocking of mobile touch screen devices by simple gestures: you can see it but you can not do it. In *International Conference on Mobile computing & networking.* ACM.

[37] Shimmer. 2018. Wearable Sensor Technology | Wireless IMU | ECG | EMG | GSR. http://www.shimmersensing.com/.

[38] Babins Shrestha, Maliheh Shirvanian, Prakash Shrestha, and Nitesh Saxena. 2016. The Sounds of the Phones: Dangers of Zero-Effort Second Factor Login based on Ambient Audio. In *Conference on Computer and Communications Security.* ACM.

[39] Prakash Shrestha and Nitesh Saxena. 2018. Listening Watch: Wearable Two-Factor Authentication using Speech Signals Resilient to Near-Far Attacks. In *Conference on Security & Privacy in Wireless and Mobile Networks.* ACM.

[40] Hossein Siadati and Nasir Memon. 2017. Detecting Structurally Anomalous Logins Within Enterprise Networks. In *Conference on Computer and Communications Security.* ACM.

[41] SONY. 2018. SmartWatch 3 SWR50. https://www.sonymobile.com/us/products/smart-products/smartwatch-3-swr50/.

[42] Pin Shen Teh, Andrew Beng Jin Teoh, and Shigang Yue. 2013. A survey of keystroke dynamics biometrics. *The Scientific World Journal* 2013 (2013).

[43] Chee Meng Tey, Payas Gupta, and Debin Gao. 2013. I can be you: Questioning the use of keystroke dynamics as biometrics. (2013).

[44] Joe Windels. 2018. How to bypass 2FA (two-factor authentication). https://www.wandera.com/bypassing-2fa/. Accessed on July 23, 2018.

[45] Junshuang Yang, Yanyan Li, and Mengjun Xie. 2015. MotionAuth: Motion-based authentication for wrist worn smart devices. In *Pervasive Computing and Communication Workshops.* IEEE.

APPENDIX

A. Interaction Signature

Figure 8 shows the acceleration generated on $\mathcal{U}$'s wrist when he interacts with the desktop. As can be seen from the figure, a K2M interaction creates distinct acceleration with a higher magnitude while a typing interaction creates the acceleration of relatively lower magnitude. When scrolling the mouse wheel, $\mathcal{U}$'s wrist remains almost stationary on the mouse that generates the acceleration with low magnitude. We also observed the similar fluctuation in the magnitude of the gyroscope signal when $\mathcal{U}$ interacts with a terminal. Thus, these three different interactions possess unique signatures that we utilize to build our Hacksaw system.

B. Keyboard Layout

We divide the keys on the keyboard into three regions – left region, middle region, and right region. Figure 9. shows the division of keys on the keyboard into these regions.

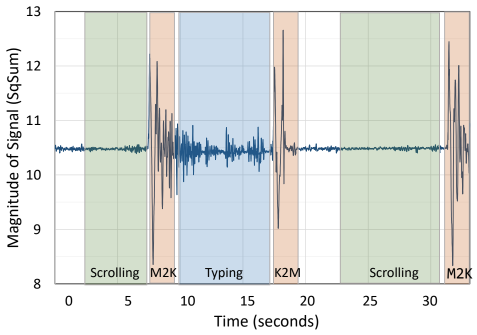


Figure 8: Acceleration of user's wrist when he interacts with a desktop.

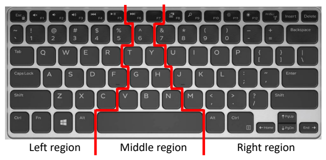


Figure 9: Keyboard layout

Table 1: List of features used in Hacksaw.

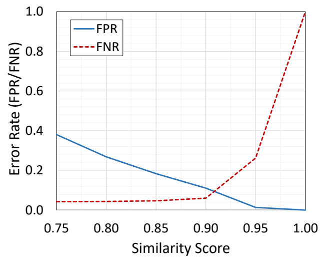
Feature	Description
Min	minimum value of signal
Max	maximum value of signal
Mean	mean value of signal
Median	median value of signal
Variance	variance of signal
Standard deviation	standard deviation of signal
MAD	median absolute deviation
IQR	inter-quartile range
Power	power of signal
Energy	energy of signal
Spectral Entropy	distribution of energy in signal
Autocorrelation	similarity of signal
Kurtosis	peakedness of signal
Skewness	asymmetry of signal
Peak-to-peak	peak-to-peak amplitude
Peak-magnitude-to-rms ratio	ratio of largest absolute value to root-mean-square (RMS) value of signal
Median frequency	median frequency of signal
Peak counts	average number of peaks and troughs per 100 ms of signal
DTW	dynamic time warping to compute similarity of inter-axis readings

C. Feature List

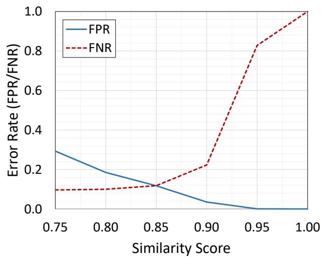
Hacksaw uses various statistical features extracted from accelerometer and gyroscope sensors when predicting interactions. Table 1 lists these features. Specifically, Feature Extractor computes a feature vector over each block of motion data received from Segmenter. Feature Extractor computes first 18 statistical features listed in Table 1 over each of the x, y, z axes readings and from magnitude ($m = \sqrt{x^2 + y^2 + z^2}$) of accelerometer and gyroscope sensor. It also uses dynamic time warping (DTW) to compute the similarity of inter-axis readings. For each of the accelerometer and gyroscope sensors, three DTW values, each corresponding to DTW of x- and y-axes readings, x- and z-axes readings, and y- and z-axes readings, are computed. Thus, Feature Extractor results in 150 (= (18∗4+3)∗2) features from each block of motion data.

D. Performance of Interaction Predictor:

Figure 10 shows FPR and FNR plots of Interaction Predictor for various interaction similarity threshold ($\theta_{interact}$).



(a) Desktop



(b) Laptop

Figure 10: FPR and FNR over different similarity score for *Interaction Predictor* of Hacksaw.

E. Hacksaw Parameters

All the parameters and their values used in Hacksaw are presented in Table 2. To find an optimal value for the parameters other than those mentioned earlier in Section 6.1, we employ the following approach. To find the minimum duration of an interaction, we analyzed all the keyboard-mouse related events in our dataset and observed that none of the interactions were less than 25 ms. So, we chose 25 ms as "Min. duration". To find the maximum duration of typing and scrolling interactions, we evaluated our approach for different durations between (500–1500) ms. Our approach performed best with 500 ms, hence we picked it as "Max. duration". For K2M, we noted the maximum duration of K2M for each participant and used their average (rounded to the nearest 100) as "Max. duration for K2M". To find an optimal value for typing threshold (v), we measured the minimum typing speed of all our participants. We used the minimum of all minimum individual typing speed as a threshold so that none of the legitimate users are incorrectly kicked out solely based on his typing speed. We found v = 2 keys/second as the typing threshold (rounded to nearest whole number) for Typing Speed Checker.

Table 2: Hacksaw Parameters.

Components	Parameters	Values
Interaction Generator	Min. duration	25 ms
	Max. duration	500 ms
	Max.duration for K2M	3500 ms for Desktop; 3000 ms for Laptop
	Idle threshold	1 s
	Similarity threshold ($\theta_{interact}$)	0.90 for Desktop; 0.85 for Laptop
Decision Module	Window size (w)	18
	Similarity threshold (θ_{win})	60%
	Bonus window (b)	1, 2
	Typing threshold (v)	2 keys/sec
	Bonus flag (f)	2