Privacy Leakage via Unrestricted Motion-Position Sensors in the Age of Virtual Reality: A Study of Snooping Typed Input on Virtual Keyboards

Yi Wu*, Cong Shi†, Tianfang Zhang‡, Payton Walker§, Jian Liu*, Nitesh Saxena§, Yingying Chen‡
*University of Tennessee, Knoxville, TN, USA
†New Jersey Institute of Technology, Newark, NJ, USA
‡Rutgers University, New Brunswick, NJ, USA
§Texas A&M University, College Station, Texas, USA
Email: ywu83@vols.utk.edu, cong.shi@njit.edu, tz203@scarletmail.rutgers.edu, prw0007@tamu.edu,
jliu@utk.edu, nsaxena@tamu.edu, yingche@scarletmail.rutgers.edu

***Abstract*—Virtual Reality (VR) has gained popularity in numerous fields, including gaming, social interactions, shopping, and education. In this paper, we conduct a comprehensive study to assess the trustworthiness of the embedded sensors on VR, which embed various forms of sensitive data that may put users' privacy at risk. We find that accessing most on-board sensors (e.g., motion, position, and button sensors) on VR SDKs/APIs, such as OpenVR, Oculus Platform, and WebXR, requires no security permission, exposing a huge attack surface for an adversary to steal the user's privacy. We validate this vulnerability through developing malware programs and malicious websites and specifically explore to what extent it exposes the user's information in the context of keystroke snooping. To examine its actual threat in practice, the adversary in the considered attack model doesn't possess any labeled data from the user nor knowledge about the user's VR settings. Extensive experiments, involving two mainstream VR systems and four keyboards with different typing mechanisms, demonstrate that our proof-of-concept attack can recognize the user's virtual typing with over 89.7% accuracy. The attack can recover the user's passwords with up to 84.9% recognition accuracy if three attempts are allowed and achieve an average of 87.1% word recognition rate for paragraph inference. We hope this study will help the community gain awareness of the vulnerability in the sensor management of current VR systems and provide insights to facilitate the future design of more comprehensive and restricted sensor access control mechanisms.**

I. Introduction

Virtual Reality (VR) technologies have been rapidly gaining popularity throughout the last decade, owing to their capability of creating an immersive environment for all users, regardless of physical constraints. A recent report reveals that the global market size of VR has grown to $21.83 billion in 2021 and will be increasing to $69.60 billion by 2028 [41]. In addition to gaming, which has been currently considered the primary usage for VR, it also brings upon innovations in a broad range of areas such as military & medical training [24], financial services [6], tourism [49], and online collaboration [37]. However, despite the great convenience VR has brought to us, its security and privacy issues have not received due attention.

Typically, a VR system consists of two types of devices: a headset that depicts the virtual world and a pair of controllers that facilitate the interaction between the user and the virtual world. Various sensors, which enable immersive human-computer interactions, are embedded in the headset & controllers to track the user's position, body movements, surroundings, and inputs. Additionally, the data recorded by these sensors unavoidably encode various types of user's private information, which introduces a severe privacy breach if they are abused by an adversary. For instance, the adversary may reconstruct the user's upper body movements; the user's gesture-based inputs to the virtual world could be potentially snooped on; and even the user's surroundings could be exposed to the adversary. While privacy leakage of different types is possible, in this work, we specifically center our focus on the following research question: *Is it possible to stealthily eavesdrop on VR sensor data, and to what extent will this information leakage expose the user's privacy?*

In this paper, we explore this question by investigating the sensor management policies in VR and provide proof-of-concept validations to demonstrate that an adversary is capable of stealthily collecting these sensor data. To further validate the severity of this privacy breach, we choose to explore this leakage risk in the context of keystroke snooping in VR, which is an important way of entering sensitive information into the virtual world and has been used in various VR domains, such as text chat, authentication password, and private healthcare/bank transaction information, etc.

Prior Research in VR Security. Existing research on VR security & privacy mainly focuses on user authentication [20], [12], [30], [33], yet the security of sensor data and the potential consequences caused by this information leakage have barely been explored. Casey *et al.* [10] proposed a set of immersive VR attacks, but their main focus is to compromise the users' safety (e.g., disorient users and modify VR environmental factors that force them into hitting physical objects) instead of compromising the users' privacy. Ling et al. [26] proposed the keystroke inference attack on a smartphone-based VR system, Samsung Gear VR, which was built on top of the Android operating system and discontinued in 2019 [19]. However, this

study only focused on a single typing mechanism for password inference and made a strong assumption that the rotation angle of the VR controller from one key to another is fixed, which would significantly reduce its attack feasibility in practice. We discuss its practicality in detail in Section IV-B. To the best of our knowledge, there has yet to be research focused on examining the security level of sensor data of more popular PC VR systems (e.g., HTC Vive Pro and Oculus).

Unrestricted Sensors in VR. We thoroughly explore the security level of various on-board sensors in the three mainstream VR Software Development Kits (SDKs) and Application Programming Interface (API), including OpenVR SDK [15], Oculus Platform SDK [45], and WebXR Device API [51]. Particularly, OpenVR and Oculus Platform SDKs have been widely employed to develop numerous VR applications, while WebXR Device API is a JavaScript application programming interface to enable applications to interact with VR devices in a web browser. For all of them, we find access to most of the sensor data does not require any user permission, creating a broad opportunity for the adversary to steal the users' private information. Leveraging the built-in functions in these SDKs/APIs, we validate that the adversary can simply deploy malware programs or fool the user into visiting malicious webpages to surreptitiously and continuously log VR sensor data in the background. These unrestricted sensor data, including the motion, position, and orientation of the headset & controller, and the button states of the controller, could expose users to serious privacy threats while using immersive VR systems.

Snooping Typed Keys on Virtual Keyboards. As an important text entry interface to enter sensitive information in VR, typing on virtual keyboards is mainly determined by the physical dynamics (i.e., position and orientation) of the controller and its button states. To show the possibility of using these unrestricted on-board sensors to snoop on keystrokes in practical attack scenarios, the adversary in our attack model is not assumed to possess any knowledge about the victim's VR setting (e.g., the placement of the stationary base stations), which determines the sensor's coordinate systems, or be able to collect any a-priori labeled training data from the victim. Specifically, our approach detects and extracts each keystroke through the unrestricted sensor data and estimates their coordinates in the 3D VR space. According to the type of the victim's inputs, we develop two sets of methods to recover passwords (random characters) and paragraphs (natural language text), respectively. Prior to the attack, the adversary reconstructs the virtual keyboard built on each key's 2D coordinate through typing with their own VR systems. Regarding the password inputs, as the victim always needs to input the *Enter* key at the end, we propose a backward inference algorithm to find potential password candidates on the reconstructed keyboard, which have the most similar trajectories compared with the user's input. As for the paragraph inputs, the adversary will first align the victim's keystrokes with the reconstructed keyboard and then employ unsupervised learning and labeling algorithms to recognize each keystroke. Although we for the first time demonstrate that the typed input in VR can be snooped via unrestricted sensors, similar keystroke inference attacks targeting mobile devices leveraging zero-permission sensors (e.g., accelerometers) have been known for years with unchanged concepts [38], [35], [9], yet the vulnerability still exists. Our main contributions are summarized as follows:

- We develop malware programs and malicious webpages to access unrestricted sensor data on the two most popular VR systems[1], and validate the severity of this privacy leakage in the context of keystroke snooping, for two interactive methods of typing (i.e., drum-based and laser-based typing), and show its possibility to recognize the user's typing with an accuracy sufficient to snoop on both passwords and natural language text.
- We develop a series of algorithms to estimate the position of the keystrokes input by the victim in a 3D space from the collected motion-position sensor data and further reveal the geometric relationship between keys to infer the victim's keystrokes. We launch the attack under a realistic but challenging scenario in which the adversary neither possess any prior knowledge about the victim's VR settings nor a-priori labeled data.
- Extensive experiments involving 14 participants and the two most popular VR systems, show that an adversary can recover the victim's password inputs with an average 84.9% recognition accuracy within three attempts, and can achieve 89.7% keystroke recognition accuracy with 87.1% WRR for natural language text.
- We discuss and analyze several potential countermeasures against this privacy breach. We hope our findings bring upon insights to the design of sensor management policies in VR and could help formulate more trustworthy immersive virtual experiences in the future.

II. Privacy Leakage Through Sensory Data in VR

A. Sensors in VR

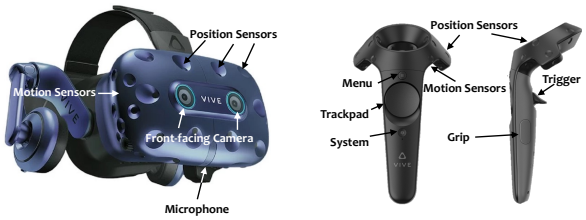
Various sensors are embedded in the headset & controllers of a VR system to enable immersive human-computer interaction in virtual environments. For instance, the headset of HTC Vive Pro, as illustrated in Figure 1 (a), contains photodetectors (i.e., position sensors) for position tracking (detailed in Section III-B), front-facing cameras that are used to capture user surroundings, a microphone that picks up user's voice input, and motion sensors (i.e., accelerometer and gyroscope) which are utilized to estimate the headset's orientation. The controller, as shown in Figure 1 (b), has multiple buttons with different functions (i.e., menu, trackpad, grip, system, and trigger), with embedded photodetectors for position tracking and built-in accelerometer and gyroscope sensors for posture estimation. Note that other VR systems, such as Oculus Quest, are equipped with a similar set of sensors to assist the user to interact with these devices in VR environments. Inevitably, the data from these sensors carry a vast amount of sensitive information that could potentially put the user's privacy at risk.

[1]Demo videos can be found at the anonymous website [2].

TABLE I
SUMMARY OF PERMISSION REQUIRED FROM USERS TO ACCESS VARIOUS SENSORS ON THE MAINSTREAM VR SDKS/API.

	Motion Sensor (Headset & Controller)	Position Sensor (Headset & Controller)	Button (Controller)	Front Camera (Headset)	Microphone (Headset)
OpenVR SDK [15]	✗	✗	✗	✓*	✓
Oculus Platform SDK [45]	✗	✗	✗	**N/A**	✓
WebXR Device API [51]	✗	✗	✗	**N/A**	✓

✓: requires permission; ✗: no permission required; and ✓*: requires global permission (no control over which app can use it)



(a) Sensors on the headset (b) Sensors on the controller

Fig. 1. Sensors in a VR system (i.e., HTC Vive Pro).

B. Sensor Management in VR

To understand the security level of these privacy-sensitive sensors, we thoroughly examine their access control management on the two mainstream VR SDKs, i.e., OpenVR SDK [15] and Oculus Platform SDK [45], and a general API for developing and hosting VR/AR on the web, i.e., WebXR Device API [51]. Unlike existing permission-based access control that helps decide if an app can access particular sensors, we find that most sensor data in VR can be easily exported without requiring explicit user permission, as illustrated in Table I. Specifically, the low-grade motion sensor data, position sensor data, and button state data can be easily accessed without requiring any permission for all the SDKs/API. Additionally, the video recorded by front-facing cameras can be accessed on OpenVR as long as the user grants global permission to its usage [10]. We find that accessing microphone data always requires user permission. However, in most cases, it remains unclear to the users whether their voice data is legally collected and used, even the user grants permission to a specific app.

C. Proof-of-Concept Validation of Stealthy Sensor Collection from Controllers

We further validate the viability of stealthily collecting sensor data (i.e., position, orientation, and button states) of VR controllers through implementing malware, which could be either running in the background of the VR environment or embedded in a VR webpage, on the two mainstream VR SDKs and the WebXR Device API. Demo videos of this validation can be found at [2]. Such a proof-of-concept validation also serves as the foundation of the attack model (Section IV-A) in snooping typing on virtual keyboards.

OpenVR SDK. We write a malware script via OpenVR [15] that continuously logs the victim's controller sensor data in the background and sends them to a remote adversarial server. As illustrated in Figure 2 (a), we use the built-in `getDeviceToAbsoluteTrackingPose` function to get the position & orientation of the controller, while the button states can be obtained via the `getControllerState` function. Both functions do not require user permission and could run stealthily in the background without being noticed by the victim. The `getDeviceToAbsoluteTrackingPose` function will return a `pose` object with the `mDeviceAbsoluteTracking` attribute, which contains the raw position data of the controller and the quaternion representation of its orientation. The `getControllerState` function will return a `VRControllerState_t` object with three critical attributes: `rAxis0`, `rAxis1`, and `ulButtonPressed`, which reflect the interaction point of the victim to the touchpad, the dynamics of the user pressing the trigger button, and whether other buttons (e.g., menu and grip) are pressed, respectively.

Oculus Platform SDK. We also build a VR program on the Oculus Platform SDK [45] (v32) based on C++ and successfully extract the aforementioned sensor data without any permission in the background. According to the Oculus Data Policy [46], we find that it's legitimate for a developer to collect the user's movement data (e.g., hand movement data) and the input button states. As illustrated in Figure 2 (b), we obtain the sensor data through the `ovr_GetTrackingState` function, which returns an object (`ovrTrackingState`) with the `HandPoses` data member containing the sensor data of the controller. Specifically, `HandPoses` includes five data attributes including `ThePose`, `AngularAcceleration`, `AngularVelocity`, `LinearAcceleration`, and `LinearVelocity`. `ThePose` records the 3D positions of the controller, and the position data is stored in a vector with three elements (i.e., positions on the *x*-, *y*-, and *z*-axes). `AngularAcceleration` and `AngularVelocity` contain the controller's angular acceleration and velocity, while `LinearAcceleration` and `LinearVelocity` include the controller's moving acceleration and velocity on the *x*-, *y*-, and *z*-axes. Moreover, the app can collect the input button states (e.g., trigger, grip, and touchpad) by calling the API function `ovr_GetInputState`, which returns states for all the controller buttons. Additionally, we also explore the potential of side-loading an Android app on Oculus to extract the sensor data, which is detailed in Appendix A.

WebXR Device API. Instead of running a script/app on the victim's desktop, we also validate that the sensor data can be eavesdropped via a malicious VR webpage leveraging the WebXR Device API [51]. WebXR enables virtual world rendering purely on a webpage and is compatible with most

```
openvr.init(openvr.VRApplication_Other)

******                                    No Permission Required
pose = vr_obj.getDeviceToAbsoluteTrackingPose(
    openvr.TrackingUniverseStanding, 0,
    openvr.k_unMaxTrackedDeviceCount),

pose_mat = pose.mDeviceToAbsoluteTracking

x, y, z = pose_mat[0][3], pose_mat[1][3], pose_mat[2][3]

yaw = 180 / math.pi * math.atan(pose_mat[1][0] /pose_mat[0][0])

pitch = 180 / math.pi * math.atan(-1 * pose_mat[2][0] /
    math.sqrt(pow(pose_mat[2][1], 2) + math.pow(pose_mat[2][2], 2)))

roll = 180 / math.pi * math.atan(pose_mat[2][1] /pose_mat[2][2])

trigger = vr_obj.getControllerState(controller_index)   No Permission Required
```

(a) Obtain sensor data from OpenVR SDK

```
ovr_GetSessionStatus(session, &sessionStatus);

assert(!sessionStatus.IsVisible);

ovr_GetPerfStats(session, &perfStats);          No Permission Required
controller_track = ovr_GetTrackingState(session, 0.0, ovrFalse);

// collect position data
pos1_x = controller_track.HandPoses[0].ThePose.Position.x;
pos1_y = controller_track.HandPoses[0].ThePose.Position.y;
pos1_z = controller_track.HandPoses[0].ThePose.Position.z;

ori1_w = controller_track.HandPoses[0].ThePose.Orientation.w;
ori1_x = controller_track.HandPoses[0].ThePose.Orientation.x;
ori1_y = controller_track.HandPoses[0].ThePose.Orientation.y;
ori1_z = controller_track.HandPoses[0].ThePose.Orientation.z;

time_sec = controller_track.HandPoses->TimeInSeconds;
```

(b) Obtain sensor data from Oculus SDK

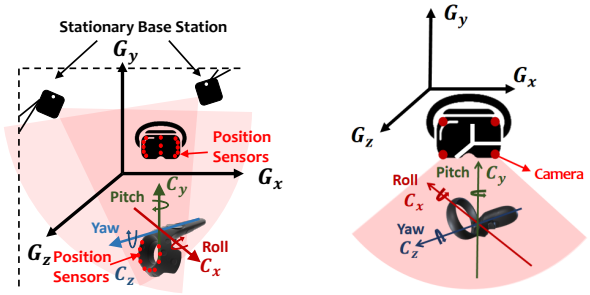
```
function onRequestSession() {
  return navigator.xr.requestSession('immersive-vr', {
      requiredFeatures: ['local-floor']
  }).then((session) => {
    xrButton.setSession(session);
    ******          Permission Required to Enter Virtual World
}
  let gripPose = frame.getPose(inputSource.gripSpace, refSpace);
    if (gripPose) {
      var x = gripPose.transform.position.x;
      var y = gripPose.transform.position.y;
      var z = gripPose.transform.position.z;
    ******                                  No Permission Required
    let gamepad = source.gamepad;
    if (gamepad) {
      if (gamepad.buttons[0].pressed) {
    ******
    }
```

(c) Obtain sensor data from WebXR Device API

Fig. 2. Code snippets of obtaining sensor data in VR on the three mainstream VR SDKs/API.

(a) Drum-based typing (b) Laser-based typing

Fig. 3. Illustration of typing in VR.



(a) Outside-in tracking (HTC Vive Pro) (b) Inside-out tracking (Oculus Quest)

Fig. 4. Position tracking & coordinate systems in VR.

browsers & VR systems. To enter the virtual world through the webpage, WebXR requires the victim's permission to create a `session` and further render the virtual world. However, after the victim enters the virtual world, there's no permission required to get the sensor data, as illustrated in Figure 2 (c). Specifically, we use the `getPose.transform` function to return a `gripPose` object of the controller. The 3D position can be obtained via the `position` attribute, and the orientation can be obtained via the `orientation` attribute. Additionally, the `inputSources.gamepad.buttons` object will reflect the controller's state, with the `pressed` attribute indicating which button is pressed by the victim.

III. Preliminaries of Typing in VR

A. Text Entry Interfaces in VR

As shown in Figure 3, users can interact with virtual keyboards via two typing mechanisms, i.e., drum-based typing and laser-based typing. Drum-based typing has been implemented for various types of applications including shopping (e.g., Vive Port [13]), designing tools (e.g., Tvori [47], Google Daydream Labs [22]), and text editors in VR (e.g., Notepad++). As illustrated in Figure 3 (a), generally the controller will be represented as a drumstick, and the user must swing the controller to hit the keys on a virtual keyboard - just like hitting drums. Differently, in laser-based typing, the controller acts like a laser pointer which allows the user to point to the key they want to enter, as shown in Figure 3 (b). A cursor reflects the intersection between the laser and the keyboard determines the key to be entered. The user then needs to press a button on the controller (usually the trigger button) to input a specific key. This way of typing has been widely deployed in various VR browsers (e.g., Firefox Reality [36]) and online meeting apps (e.g., Vive Sync [14], VRChat [23]). We further examine the typing mechanisms and keyboard layouts of 13 apps selected from top sellers in Vive Port [4], Oculus Store [3], and top VR apps in 2022 [1]. As listed in Table II, we find that all of these apps either apply drum-based typing or laser-based typing. Given the popularity of these two typing mechanisms, we target both of them in our attack scenarios. In addition, we also examine their virtual keyboard layouts and find that they all use the standard QWERTY keyboard for alphabets, and most keyboards have numeric keys appearing on the top with *Enter* on the right. As these keyboards share a similar layout, the keystroke inference attacks relying on the relative positions among keys are transferable across different virtual keyboards in VR. This also validates the generality and severity of the proposed privacy leakage attack.

B. Position Tracking & Coordinate Systems

To show our attack's generality, we consider the following two major types of visual position tracking systems:

Outside-in Tracking. Outside-in tracking uses visual sensors (e.g., cameras or laser-sensors) placed in a stationary location and oriented towards the VR headset and controllers to track their positions. Various VR systems, such as HTC Vive Pro, Oculus Rift, and PlayStation VR, use this type of tracking. Figure 4 (a) illustrates the setting of the outside-in tracking used by HTC Vive Pro. Specifically, two base stations, which are fixed within the environment, continuously emit infrared (IR) laser beams across the space, while the position sensors embedded in the headset & controllers measure the timing of laser sweeps and further estimate the relative position. The estimated positions of the headset and controllers are in the same global VR coordinate system G, which is determined by the placement of the two base stations. In addition, the headset & controllers also have their local coordinate systems used for

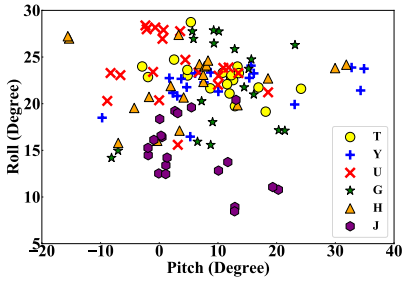


Fig. 5. Orientation of the Controller while Typing.

deriving rotation angles. For instance, the embedded motion sensors can return the orientation (i.e., pitch, roll, and yaw) of the controller with respect to its local coordinate C, as shown in Figure 4 (a).

Inside-out Tracking. Unlike outside-in tracking, inside-out tracking uses multiple cameras on the headset to observe visual features (e.g., visual patterns of furniture and devices) in the environment and triangulate the headset's and the controllers' 3D positions. In addition, inside-out tracking determines the relative position of the controllers to the headset with infrared (IR) LEDs on the controllers (e.g., IR rings on the 2nd generation of Oculus Touch). Real-time motion sensor data from the headset and the controllers (e.g., accelerometer and gyroscope readings) are leveraged to continuously refine the estimated 3D positions. As such a tracking scheme does not require to install base stations in the environment, it has been widely used in many latest headsets, including Oculus Quest 1&2, Vive Cosmos, and Samsung HMD Odyssey. Similar to the outside-in systems, the headset and the controllers are in the same global VR coordinate system G for the position tracking, which is initialized when the VR devices are turned on, as illustrated in Figure 4 (b). The rotational motion vectors of the headset & controllers are with respect to their local coordinate systems.

IV. Threat Model & Attack Overview

A. Threat Model

We consider an attack scenario in which an adversary seeks to infer a victim's keystrokes on the virtual keyboard through the VR controllers. We assume the adversary can fool the victim into either installing a malware program developed on the mainstream VR SDKs (e.g., OpenVR SDK or Oculus SDK), or visiting a malicious VR webpage created by WebXR Device API. In either case, the adversary can obtain the sensor data of the victim's controller remotely by leveraging the built-in functions described in Section II, including the controller's position, orientation (i.e., pitch, yaw, and roll), and the button states (i.e., which button is pressed). The malware program can be either disguised as a legitimate VR app/plugin, or embedded into a third-party VR development library, which might be used by many developers to build their VR apps. For instance, the adversary can embed the malicious code snippet in a seemingly benign software, such as a PC cleaning software or a system monitoring tool. The adversary can also post the malware online and lure the victims to install it by themselves. As these types of software tend to work in the background and can run along with other processes, it can stealthy log the sensor data while the victim is typing in VR, which is hard to be noticed. Compared to the malware program, attacks by hosting a malicious VR webpage would be more accessible to the adversary and more devastating as it does not require installing a third-party app on the victim's device.

Note that, in this paper, we do not consider the scenario in which the keystrokes can be directly accessed by the malware. We thoroughly examined the functions in the four SDKs; however, there are no functions that can directly access the victim's keystroke inputs from the VR app. Since the keystrokes are stored in the app's local variables, directly accessing them is impossible without modifying the app. Adding malicious functions to the SDKs or directly modifying the app would make the assumption too strong. Instead, the adversary chose to leverage the zero-permission sensor data as the side-channel to launch the attack.

In our considered attack model, the adversary doesn't possess any knowledge about the victim's VR system setting, such as the placement of the stationary base stations, which determines the devices' position tracking coordinates. The adversary doesn't need to collect any a-priori labeled training data from the victim either. We only assume the adversary has prior knowledge about the virtual keyboard that the victim uses, including the information about the virtual keyboard layout and typing mechanism (i.e., drum-based or laser-based typing). The adversary can then leverage his/her own VR device in any settings to build a reference pattern for the attack. We further discuss the scenario in which the adversary does not possess this knowledge and uses a completely different keyboard to launch the attack in Appendix G. Although this knowledge is not directly known to the adversary in practice, the adversary can attempt to use various typing interfaces based on their popularity and estimate this knowledge by checking the intelligibility of the identified text inputs. As concluded in Table II, all keyboards in these mainstream applications employ the same QWERTY layout for alphabets, with most of them having 10 numeric keys on the top of the keyboard and *Enter* on the right side, making this knowledge can be easily estimated by the adversary even if the layout of the keyboard is unknown. Additionally, only laser-based typing needs to press the button on the controller for each keystroke, thus the adversary can use the button status to identify whether it is drum-based typing or laser-based typing. We believe that the proposed attack is under a very practical threat model and can be surreptitiously and easily launched in practice.

B. Challenges & Assumption Validation

Challenges. Unlike physical keyboards, virtual keyboards usually have unfixed positions, postures, and scale sizes in the virtual environment, which are dominated by the virtual scenes in which the user is located and the user's orientation when typing function is called. In addition, the position sensor readings from the VR devices are all with respect to their global VR coordinate system (G in Figure 4), which is dependent on the device settings, such as the placement positions

of the two stationary base stations and the position where the devices are initialized/turned-on. With different virtual keyboard positions & orientations and coordinate systems, the motion-position data of the adversary and victim will exhibit completely different patterns, making the adversary unable to directly predict the victim's keystrokes from the sensor data. The adversary needs to accurately estimate the position of the keystrokes typed by the victim and further align the victim's coordinate system with his/hers to reveal the geometric relationship between keys, which poses a great challenge if the adversary neither possesses any prior knowledge about the victim's VR settings nor a-priori labeled training data.

Assumption Validation. A recent study [26] has shown the possibility of using motion sensors to snoop on keystrokes on a smartphone-based VR system, Samsung Gear VR. However, it only targets laser-based typing and assumes that the controller's rotation angles while typing the same key are highly consistent and the angles remain distinguishable while typing different keys. However, in practical typing scenarios, users tend to move around the controller frequently, which makes it hard for the user to remain the controller at the same position while typing on virtual keyboards, even on the same key. As the controller's orientation to a specific key is highly dependent on the controller's position, it is impractical to assume each key has distinguishable rotation angles of the controller. To validate this, we ask one participant to repeatedly type six adjacent keys (i.e., T, Y, U, G, H, J) and the "Enter" key on a laser-based keyboard using HTC Vive Pro. Figure 5 illustrates the rotation angles of each keystroke relative to the "Enter" key along the *x*-axis and *y*-axis in Figure 4 (i.e., roll and pitch). It is clear to observe that the rotation angles for each key are highly inconsistent, which demonstrates that the assumption made in the prior work cannot be generalized in practical typing scenarios.

C. Attack Overview

The goal of our attack is to snoop on keystrokes on virtual keyboards leveraging the unrestricted sensors of current VR systems. We consider an attack scenario, where the victim uses either drum-based typing or laser-based typing to enter inputs, which could be natural language or passwords composed of random characters. As illustrated in Figure 6, to launch attacks, the adversary can stealthily collect the sensor data associated with the victim's typing from the VR device (i.e., position, orientation, and button states of the VR controller) through the deployed malware programs or malicious webpages (Section II-C). The adversary then recognizes its typing mechanisms (i.e., drum-based or laser-based typing) and detects each keystroke segment. According to the typing mode, we design two approaches to estimate each typed key's 3D position in the virtual environment, namely, *3D Keystroke Position Estimation* and *3D Cursor Position Estimation*. The adversary then employs *Keyboard Plane Estimation & 2D Keystroke Position Projection* to identify the 2D plane of the virtual keyboard and project all of the 3D keystrokes to the plane to get their corresponding 2D coordinates.

As the adversary doesn't possess any knowledge about the virtual keyboard used in the victim's VR environment, the adversary will need to type each key of the keyboard using their own VR devices to reconstruct a 2D keyboard. Although the reconstructed virtual keyboard might be different from the one victim uses in terms of their positions and postures within their own coordinate systems, their key-to-key geometric relationship should be highly consistent. Thus, the 2D keyboard reconstructed by the adversary's typed data will serve as a reference to help recognize the victim's typing.

According to input length and the last entered key (the last key entered in the password input should be the "Enter" key), the adversary can determine whether the victim has entered a password or natural language text. If the victim has entered a password, the adversary will perform *Password Inference via Tree-based Backward Typing Trajectory* to inversely infer the password sequence from the "Enter" key and generate a set of password candidates. The adversary can then rank these candidates through analyzing the angles and distances of typing trajectory in *Ranking Password Candidates via Typing Path Analysis*. As for the natural language input, the adversary will first cluster all the detected keystrokes in an unsupervised manner via *Keystroke Clustering via DBSCAN [16]*. The centroids of the key clusters formulate the keyboard used by the victim. The adversary then performs *Keyboard Alignment via LSE [25]* to align the victim's keyboard with the 2D keyboard reconstructed by the adversary and further recognizes each keystroke leveraging *Keystroke Labelling via KNN*. To further improve the typing reconstruction, the adversary will adopt language models to fix the grammatical & spelling errors in the reconstructed sentences.

V. Attack Design

A. Typing Mechanism Recognition & Keystroke Detection

In our attack, we consider an input session in which the victim continuously types on the virtual keyboard in VR. Drum-based typing requires the user to swing the controller to hit keys, while laser-based keystroke is triggered by the controller's button. Thus, the adversary can use the button state and its pressing frequency to detect laser-based input sessions. Specifically, the adversary can detect the input session through solving the following equation:

$$\begin{aligned} &\arg\max_{t_s,t_e} t_e - t_s, \\ s.t., f_{min} < freq(t_s, t_e) < f_{max}\,, &t_{min} < t_e - t_s, \end{aligned} \tag{1}$$

where t_s, t_e are the timestamps when the victim starts/ends the input session, t_{min} is the minimal duration of typing, $freq(t_s, t_e)$ is the frequency of the trigger button being pressed within $[t_s, t_e]$, f_{min} and f_{max} are the lower/upper bounds of $freq(t_s, t_e)$. According to the Word Per Minute (WPM) for laser-based typing [8], f_{min} and f_{max} are set to 1 Hz and 2 Hz, respectively. t_{min} is set to 5 seconds to make the input session remain a reasonably long time period. If Equation 1 can be solved, all keystrokes can be easily detected within the input session, while other button activities outside

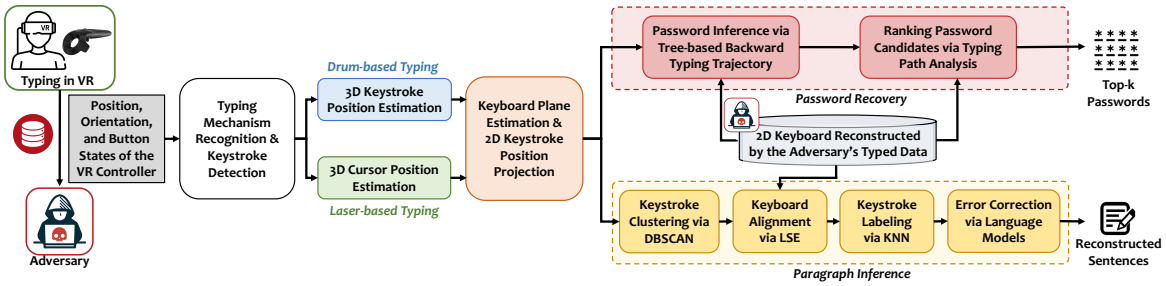
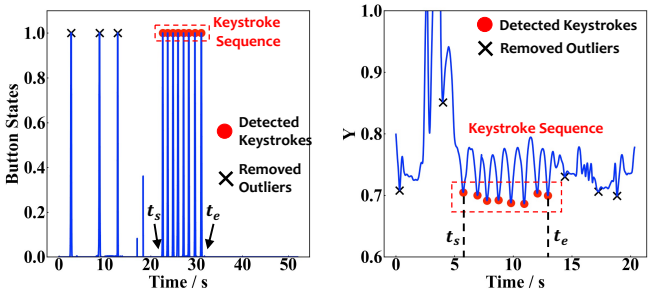


Fig. 6. Attack Overview.



(a) Laser-based typing (b) Drum-based typing

Fig. 7. Illustration of Keystroke Detection.

the session would be considered as outliers (non-typing-related button activities), as illustrated in Figure 7 (a).

As for drum-based typing, swinging the controller (like hitting drums) will cause significant displacement vertically, which will reflect on the G_y-axis of the position data. Thus, a valley detection algorithm provided by the Scipy toolkit [48] can be used to detect valleys along the G_y-axis. As the WPM for drum-based typing ranges from 13.43 to 29.81 [8], the adversary regulates the distance between two adjacent valleys to at least 0.4 seconds. The prominence of the valley, which is the vertical distance between the valley and its highest contour line, is empirically set to 3 cm. The adversary then uses Equation 1 to detect the input session t_s and t_e, but f_{max} is set to 2.5 Hz according to the WPM of drum-based typing, and $freq(t_s, t_e)$ is the frequency of the valleys that appear in the input session. An example of the detected keystrokes and input session for drum-based typing is illustrated in Figure 7 (b).

Effectiveness Validation. To validate the effectiveness of the detection algorithm, we collect 3-hour data from three participants using HTC Vive Pro, with each participant including 30-minutes data for both laser-based typing and drum-based typing. For each typing mechanism, each participant is asked to type 10 sentences in Table III with a total number of 455 characters, then conduct three different types of activities in the remaining minutes: playing a VR game (i.e., Beat Saber), browsing immersive websites, or watching an immersive video. We use True Positive Rate (TPR) and False Positive Rate (FPR) as evaluation metrics. Specifically, TPR is the ratio of correctly detected keystrokes among all keystrokes, and FPR is the ratio of falsely detected keystrokes among all outliers (e.g., button activities & valleys caused by non-typing-related activities). For drum-based typing, we reach 97.9% TPR and 5.7% FPR, while the TPR and FPR are 99.4% and 5.3% for laser-based typing, respectively. These promising results demonstrate that we can successfully distinguish typing activities from other irrelevant button activities & body movements.

B. Keystroke Position Estimation

Based on the typing mechanism, the adversary can estimate each keystroke's position using the following methods:

3D Keystroke Position Estimation. In drum-based typing, the positions of each entered key can be directly obtained from the controller's positions. Particularly, the adversary uses a window of 0.5 seconds centered at the detected valley along the G_y-axis to calculate the position mean along each axis in the frame G. Each keystroke can be then represented as a 3D vector containing its position in the 3D VR space.

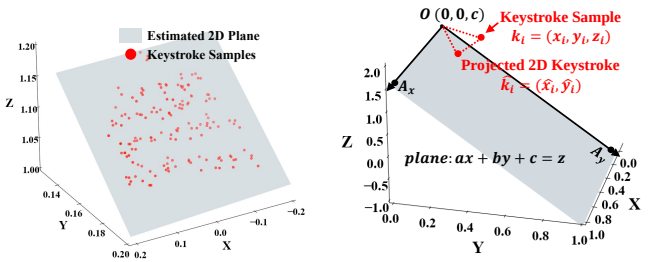
3D Cursor Position Estimation. In laser-based typing, the key entered by the laser pointer is dependent on both the position and orientation of the VR controller. As the laser is in the direction of the C_z-axis of the controller, its direction is determined by the pitch (α) and roll (β) of the controller and is irrelevant to the yaw angle (Figure 4). The rotation matrix $\mathbf{R}$ of the laser, with respect to its initialized stage where pitch & roll are all zero, can thus be derived as:

$$\mathbf{R} = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) \\ 0 & \sin(\beta) & \cos(\beta) \end{bmatrix}. \quad (2)$$

To derive the position of the cursor after rotation, the adversary needs to know its position at the initialized stage (i.e., pitch and roll equal to zero). Although the adversary doesn't possess any information about the global coordinate system G of the victim, we observe that the controller's pointing direction (i.e., C_z-axis) is always initially aligned in the negative direction of the G_z-axis. Given the position of the controller as $[x, y, z]^T$, the position of the cursor P_c on the virtual keyboard can thus be estimated as $P_c = \mathbf{R} \cdot [x, y, z - l]^T$, where l is the distance between the keyboard and the controller along the z-axis, which is relatively consistent and can be easily estimated by the adversary with his/her own typed data.

C. Keyboard Plane Estimation & 2D Keystroke Position Projection

As all keys are located on the same plane (i.e., the plane of the virtual keyboard), the adversary's next step is to identify this 2D plane and project all the keystrokes to it to get their 2D



(a) Keyboard plane estimation (b) 2D keystroke position projection

Fig. 8. Illustration of Keyboard Plane Estimation & 3D-to-2D Projection.

coordinates. Specifically, we consider the virtual keyboard's plane as $ax+by+c = z$, and the i_{th} keystroke's 3D coordinate is represented as (x_i, y_i, z_i). Since all keystrokes are on the same plane, we can get the following equation:

$$\begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ & \dots & \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ \dots \\ z_n \end{bmatrix}, \tag{3}$$

where n is the number of detected keystrokes. Equation 3 can be represented as $\mathbf{AX} = \mathbf{B}$ for simplicity. The unknown vector $\mathbf{X}$ can be derived using Least Square Estimation (LSE) [25]:

$$\mathbf{X} = \begin{bmatrix} a & b & c \end{bmatrix}^T = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{B}. \tag{4}$$

Figure 8 (a) depicts 100 keystrokes in the 3D VR space with the identified 2D plane of the virtual keyboard.

To obtain the 2D coordinates of the keystrokes projected onto the plane, we define $O = (0, 0, c)$ as the origin of the 2D plane, as shown in Figure 8 (b). $A_x = (1, 0, a + c)$ is in the plane's positive x-axis direction. To determine the direction of the y-axis, we define a point $A_y = (x_y, 1, z_y)$ on the y-axis and solve the following equations:

$$\begin{cases} ax_y + b + c = z_y \\ \mathbf{A_x} \cdot \mathbf{A_y} = 0 \end{cases} \tag{5}$$

where $\mathbf{A_x}$ and $\mathbf{A_y}$ are the vectors from O to A_x and A_y, respectively. After we obtain A_y, which is $(-\frac{ab}{a^2+1}, 1, \frac{b}{a^2+1} + c)$, the 2D coordinate $(\hat{x_i}, \hat{y_i})$ of the keystroke i in the plane can be obtained using the following equation:

$$\hat{x_i} = \frac{\mathbf{k_i} \cdot \mathbf{A_x}}{||\mathbf{A_x}||}, \hat{y_i} = \frac{\mathbf{k_i} \cdot \mathbf{A_y}}{||\mathbf{A_y}||}, \tag{6}$$

where $\mathbf{k_i}$ is the vector from the origin O to the i_{th} keystroke's 3D coordinate (x_i, y_i, z_i).

D. 2D Keyboard Reconstructed by the Adversary's Typed Data

Prior to inferring the victim's keystrokes, the adversary will first generate a reconstructed keyboard through typing with his or her own VR system. Specifically, the adversary will repeatedly type each key multiple times and then process the sensor data via the aforementioned detection, position estimation, and projection mechanisms without too much effort. To further reduce the required effort, it is possible to use fewer keys to reconstruct the keyboard through mapping them to the standard QWERTY layout. The adversary further implements the K-means clustering algorithm [31] on the projected 2D keystrokes, where k is set to the number of keys being typed. The detected centroids will formulate the reconstructed keyboard, which will be further used as a reference to help infer the typed inputs of the victim.

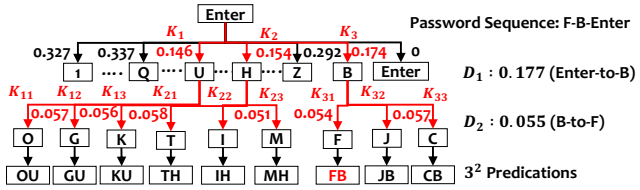
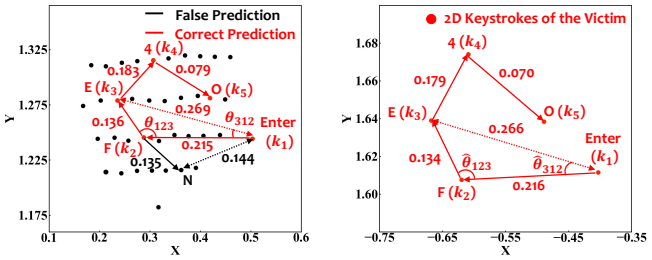


Fig. 9. Tree-based Backward Typing Trajectory Estimation for Password Recovery.



(a) Inferred trajectory on the reconstructed keyboard (b) Trajectory of the victim's typing

Fig. 10. Backward Password Inference Leveraging Accumulated Displacement & Orientation.

E. Password Recovery

Password Inference via Tree-based Backward Typing Trajectory. As the victim will always input the "Enter" key at the end, the adversary employs a tree-based backward inference algorithm leveraging on the reconstructed 2D keyboard. An example of inferring a 2-character password is illustrated in Figure 9. Specifically, the "Enter" key in the reconstructed keyboard serves as the root node of the tree, and the adversary calculates its absolute distance to all other keys in the reconstructed keyboard. The adversary then compares these distances with the absolute distance D_1 between the last key (i.e., the *Enter*) and the second-to-last key (i.e., the last character of the password) of the victim. The adversary finds the top 3 keys K_1, K_2, K_3, which are closest to D_1. The adversary then increases the depth of the tree and repeats the same step for K_1, K_2, K_3 and find another set of three closest keys (e.g., K_{11}, K_{21}) in the next layer of the tree for each of them. The same step is repeated for the newly added keys until the depth of the tree increases to the length of the password n, at which point the adversary can finally obtain 3^n possible password candidates.

Ranking Password Candidates via Typing Path Analysis. The tree-based inference algorithm only considers the distance between adjacent keys, which will yield many false predictions. For instance, Figure 10 shows an example of inferring the "O-4-E-F-Enter" sequence using the reconstructed keyboard. We observe that the L2 distance between "E" and "F" in the victim's trajectory is 0.134 in Figure 10 (b). In the process of backward inferring the next character of

"Enter-F", the adversary may mistakenly predict it as "N" rather than "E", because in the reconstructed keyboard the distance between "N" and "F" (i.e., 0.135) is closer to 0.134 compared with the distance between "E" and "F" (i.e., 0.136). To quantify the errors between the inferred password path on the reconstructed keyboard and the victim's actual typing path, the designed method takes into account both accumulative distance and orientation similarity of the typing path to sort all the password candidates derived from the tree-based backward password inference. Specifically, for the n_{th} candidate, and for every possible keystroke pair $\{k_i, k_j\}$ (i.e., i_{th} and j_{th} keystrokes), the adversary calculates the relative difference of their L2 distances between the reconstructed keyboard $D(k_i, k_j)$ and the victim's trajectory $\hat{D}(k_i, k_j)$ and further adds them together to get the accumulated distance $\mathbb{D}_n$:

$$\mathbb{D}_n = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{|D(k_i, k_j) - \hat{D}(k_i, k_j)|}{\hat{D}(k_i, k_j)}. \quad (7)$$

Additionally, the adversary also calculates the orientation similarity between a candidate and the trajectory of the victim leveraging the intersection angles formulated by three different keystrokes, as illustrated in Figure 10. Given a combination of three keystrokes $\{k_i, k_j, k_k\}$, the intersection angle θ_{ijk} can be calculated as:

$$\theta_{ijk} = arccos(\frac{\overrightarrow{k_i k_j} \cdot \overrightarrow{k_j k_k}}{|\overrightarrow{k_i k_j}||\overrightarrow{k_j k_k}|}). \quad (8)$$

Similar to the accumulated L2 distances, the adversary accumulates the relative difference of these angles between the reconstructed keyboard and the victim's trajectory to get the orientation difference $\mathbb{O}_n$ (Note $\theta_{ijk} = \theta_{kji}$):

$$\mathbb{O}_n = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=i+1}^{n} \frac{\theta_{ijk} - \hat{\theta}_{ijk}}{\hat{\theta}_{ijk}} (i \neq j \neq k). \quad (9)$$

The relative error for the n_{th} candidate $\mathbb{E}_n$ is thus defined as $\mathbb{D}_n + \mathbb{O}_n$. The adversary then sorts all the password candidates based on it in ascending order.

No-Enter Key Scenario. If the "Enter" key is not pressed at last, given the password length as *n*, the adversary treats each possible combination that contains *n* keys as a password candidate. The adversary then performs the same typing path analysis algorithm to analyze the relative error between the candidate and the keystrokes input by the victim, then sort all the candidates based on the error in ascending order.

F. Paragraph Inference

Keystroke Clustering via DBSCAN. For natural language inputs, as the positions of keystrokes for the same key are very close to each other, the adversary employs DBSCAN [16], a density-based spatial clustering algorithm, on the projected 2D keystrokes. The minimum number of instances in each cluster is set to 2, and the maximum distance between two instances that can be considered as neighbors in the same cluster is empirically set to 0.03, which is approximately the average distance between adjacent keys on the virtual keyboard.

Keyboard Alignment via LSE. Since the victim's keyboard and the adversary's reconstructed keyboard are usually generated under different coordinate systems, their position, orientation, and scale may differ a lot. Nonetheless, the keyboard layout should be relatively consistent. Thus, in this step, the adversary can align the keyboard of the victim to the adversary's reconstructed keyboard using LSE. Specifically, the adversary randomly selects n keys from the reconstructed keyboard, where n is the number of detected DBSCAN clusters. Given the coordinates of the i_{th} detected centroids as $[x_i, y_i]$ and the i_{th} selected key in the reconstructed keyboard as $[\hat{x}_i, \hat{y}_i]$, the following equation can be obtained:

$$\begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ & \dots & \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} = \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \dots \\ \hat{x}_n \end{bmatrix}, \begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ & \dots & \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} = \begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \dots \\ \hat{y}_n \end{bmatrix}, \quad (10)$$

where $a_1, b_1, c_1, a_2, b_2, c_2$ can be solved using Equation 4. The ith key on the aligned keyboard $[\hat{x}'_i, \hat{y}'_i]$ can thus be calculated as $[a_1 x_i + b_1 y_i + c_1, a_2 x_i + b_2 y_i + c_2]$. To find the most appropriate alignment, the adversary uses the average distance between $[\hat{x}'_i, \hat{y}'_i]$ and $[\hat{x}_i, \hat{y}_i]$ as the metric.

Keystroke Labelling via KNN & Error Correction via Language Models. After aligning the victim's keyboard to the adversary's reconstructed keyboard, the adversary can simply use the KNN classification algorithm to recognize each keystroke typed by the victim. The reconstructed keyboard serves as the training data, and K is set to 1. To further improve the reconstructed natural language text, the adversary will use a language tool to fix the grammatical & spelling errors for a more precise prediction. We choose to use the "spelling and grammar" function in Google Docs, yet many other online language tools can be used as well.

VI. Attack Evaluation

A. Experimental Methodology

Devices & Virtual Keyboards. We evaluate the proposed attack using two mainstream VR systems: HTC Vive Pro and Oculus Quest, which belong to outside-in and inside-out tracking, respectively. HTC Vive Pro is connected to an Alienware desktop running on Windows 10 with a GeForce GTX 1660Ti GPU, while Oculus Quest is connected to a Lenovo Legion 5 laptop equipped with a GeForce RTX 2060 GPU running on Windows 10. The developed malware programs (Section II-C) have been installed in these computers so that the adversary could remotely access the unrestricted sensor data. We use both drum-based and laser-based keyboards. Specifically, for HTC Vive Pro, we chose the built-in keyboards of Tvori (drum-based) and Vive Sync (laser-based), while for Oculus Quest, we use the virtual keyboard of Notepad++ for both drum-based and laser-based typing styles. By default, the sensor data is sampled at 250 Hz for HTC Vive Pro and 60 Hz for Oculus Quest.

Typing Data Collection. Our data collection involves 7 participants for each VR system (14 participants in total). The participants include 11 males and 3 females, aging from

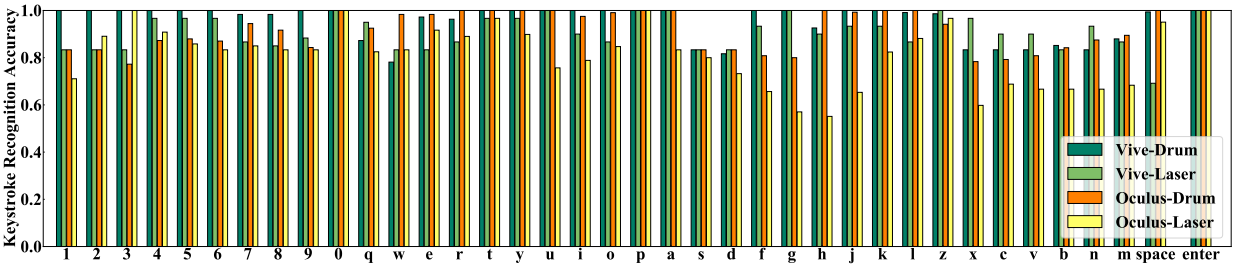
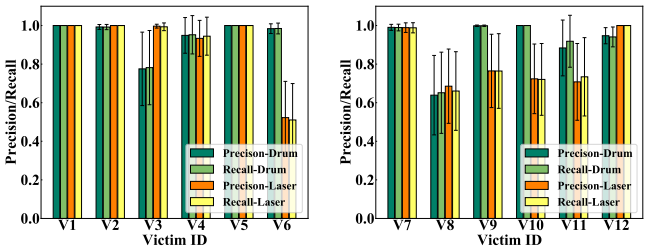


Fig. 11. Performance of Single Keystroke Recognition.



(a) HTC Vive Pro (b) Oculus Quest

Fig. 12. Performance of Keystroke Recognition for Drum-based & Laser-based Typing on HTC Vive Pro & Oculus Quest.

22 to 34. The participants are asked to type on the virtual keyboards in VR for some time to get familiar with VR typing before the official data collection. For each VR system, one of the participants pretends to be the adversary and the remaining six participants are treated as victims. We set different VR systems' settings (e.g., positions of the stationary base stations used for HTC Vive Pro) between the adversary and victims to create a more realistic attack scenario. For each virtual keyboard, all participants (both the adversary and the victims) are asked to type 38 keys (i.e., 26 alphabets, 10 numeric, space, and enter) repeatedly, 20 times each. The typed data of the adversary will be utilized to generate the reconstructed keyboard (Section V-D), and the performance for single keystroke recognition is evaluated in Section VI-B. We randomly generate three different passwords with lengths of 4, 6, and 8 for each victim. For each key in a password, we randomly select one keystroke that stands for the specific key from the victim's typed data, with a randomly chosen"Enter" key at last. These randomly selected keystrokes formulate a password input, and we formulate 42 different passwords for a more comprehensive evaluation. We evaluate the performance of the password recovery attack in Section VI-C. Additionally, all victims are asked to type 10 randomly selected phoneme balanced sentences from the Harvard sentences dataset [42] in Table III. The results of inferring these sentences using the attacker's typed data are presented in Section VI-D. We also conduct the same experiments on an Android app, and the performance evaluation is detailed in Appendix E. We also extend the password recovery attack through involving no-enter key scenario and upper cases & symbols and , which is detailed in Section VI-E and Appendix F, respectively. Furthermore, we discuss the worst-case scenario, in which the adversary and the victim use completely different keyboards, in Appendix G. During data collection, we let the participants equip the headset on their own and do not control their standing position, facing orientation, or typing speed for any of the experiments. Each participant is asked to conduct experiments in different sessions, and their WPM varies from 15.2 to 21.7 for drum-based typing, and 11.3 to 17.8 for laser-based typing. Additionally, we do not collect any labeled data from the victims prior to the attack. In total, we collect 3,480 keystrokes from the two adversaries and 36,880 keystrokes from the 12 victims in a one-year time period. The data collection procedures were approved by our university's IRB through an expedited review procedure.

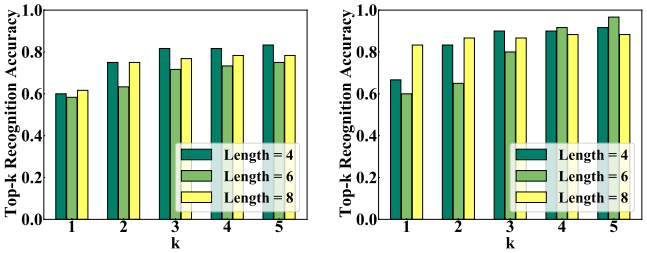
Evaluation Metrics. *Accuracy, Precision* and *Recall* are used to evaluate single keystroke recognition. The accuracy for the key k is defined as the percentage of the keystrokes that are correctly classified as k among all keystrokes of k. The precision of the key k is defined as $\frac{TP_k}{TP_k+FP_k}$ and the recall of the key k is defined as $\frac{TP_k}{TP_k+FN_k}$, where TP_k, FP_k, FN_k are the true positive rate, false positive rate, and false negative rate for the key k, respectively.

Top-k Recognition Accuracy is used to evaluate the password recovery attack. Since our algorithm will return a number of potential candidates in a descending order based on the accumulated distance & orientation similarity, *Top-k Success Rate* is defined as the probability of the first k candidates containing the password input of the victim.

Word Recognition Rate (WRR) is used to evaluate paragraph inference attack. WRR is the ratio of correctly recognized words to the total number of words typed by the victims.

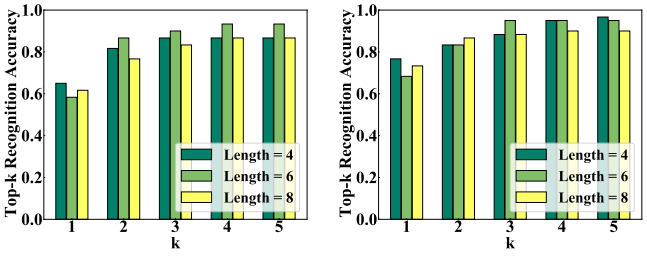
B. Performance of Single Keystroke Recognition

The single keystroke recognition performance for each key is illustrated in Figure 11. We find that HTC Vive Pro has prominent performance under both drum-based typing and laser-based typing, most of which can reach over 90% recognition accuracy with an average of 95.2% and 90.8% accuracies, respectively. Although the overall performance is slightly lower for Oculus Quest, we can still reach more than 60% accuracy for most keys, with an average accuracy of 91.7% for drum-based typing and 81.1% for laser-based typing. Additionally, the precision/recall scores for each victim are shown in Figure 12. For HTC Vive Pro, most precision and recall scores of both drum-based and laser-based typing are over 90.0%. For Oculus Quest, the total performance is a bit lower but the precision/recall scores among most users are still over 75.0%. The results demonstrate the effectiveness of the proposed method on single keystroke recognition.



(a) HTC Vive Pro (b) Oculus Quest

Fig. 13. Performance of Recovering Passwords on Drum-based Typing.



(a) HTC Vive Pro (b) Oculus Quest

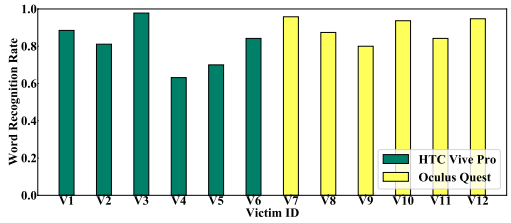
Fig. 14. Performance of Recovering Passwords on Laser-based Typing.

We find that the "Enter" key can always achieve very high accuracy, which is highly likely due to its 'isolated' characteristics. We have also noticed that for some keys, especially for laser-based typing on both HTC Vive Pro and Oculus Quest, the accuracy is less than 60% (e.g. 's' for HTC Vive Pro and 'x' for Oculus Quest). We further look into the misclassification results, and find that in most cases the false prediction tends to be the neighboring keys of the ground truth. However, we find this type of error is relatively insignificant and can be easily corrected, detailed in Section VI-D.

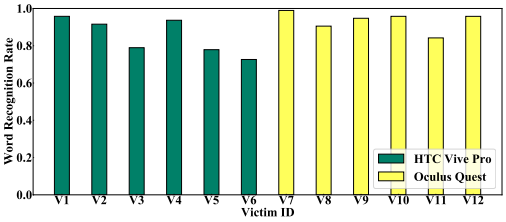
C. Performance of Password Recovery

We further evaluate the designed attack under more practical attack scenarios of deriving passwords. Figure 13 (a) and (b) show the accuracy of inferring passwords with three different lengths on HTC Vive Pro and Oculus Quest for drum-based typing, respectively. We find that for HTC Vive, our attack can achieve (60.0%, 58.3%, 61.7%) top-1, (81.7%, 71.7%, 76.8%) top-3, and (83.3%, 75.0%, 78.3%) top-5 recognition accuracies on inferring passwords with 4, 6, and 8 keys, respectively. For Oculus Quest, we find that the attack has much better performance, with (66.7%, 60.0%, 83.3%) top-1, (90.0%, 80.0%, 86.7%) top-3, and (91.7%, 96.7%, 88.3%) top-5 recognition accuracies. An encouraging finding is that Oculus Quest can achieve close to 85.0% top-1 success rate for the key length of 8, indicating that longer key length may lead to more severe password leakage. In addition, both headsets have over 71.7% top-3 recognition accuracies for passwords of all three lengths.

We also evaluate the password inference performance for laser-based typing on both VR systems. The results are shown in Figure 14 (a) and (b). We find that the attack achieves high performance on Oculus Quest, with (76.7%, 68.3%, 73.3%) top-1, (88.3%, 95.0%, 88.3%) top-3, and (96.7%, 95.0%, 90.0%) top-5 recognition accuracies. We have similar



(a) Drum-based typing



(b) Laser-based typing

Fig. 15. Performance of Paragraph Inference Attack.

observations on inferring passwords with 4, 6 and 8 keys on HTC Vive, and the attack can achieve (65.0%, 58.3%, 61.7%) top-1, (86.7%, 90.0%, 83.3%) top-3, and (86.7%, 93.3%, 86.7%) top-5 recognition accuracies. Both headsets have over 86.7% top-5 success rate under laser-based typing. We observe that the length of the password only has a subtle influence on the success rate, which indicates the robustness of our proposed attack.

D. Performance of Paragraph Inference

Figure 15 illustrates the WRR of recovering the sentences listed in Table III that are typed by the victims. We observe that our attack can accurately recover the language text typed by the victims. Specifically, the average WRRs of HTC Vive Pro are 80.8% and 85.1% for drum-based typing and laser-based typing, respectively. The WRRs of Oculus Quest for drum-based typing and laser-based typing are 89.3% and 93.3%, respectively, which are even higher than HTC Vive Pro. Some examples of the recovered paragraphs are shown in Figure 16. Before error correction, we observe that there are only 1-2 false character predictions in a single word in most cases, and almost all of them are caused by false predictions of the correct key as one of its adjacent keys (e.g., recognize 'o' as 'i' or 'i' as 'u'). However, this type of error is commonly considered as typos or spelling & grammatical errors by online language tools and thus can be easily corrected. Specifically, the language tool can increase the WRR by 25.9% and 22.4% for drum-based typing and laser-based typing, respectively. The average WRR among all victims and both typing mechanisms is 87.1%, which demonstrates the effectiveness of our attack on recognizing natural language text inputs of the victim.

E. Impact of the Enter Key

We further evaluate the password recovery attack in the "No-enter Key" scenario. Same with the aforementioned methodology, we randomly generate passwords with lengths of 4, 6, and 8 for each victim, but the "Enter" key is not involved at last. We then use the brute force attack described in Section V-E to infer the generated passwords. The results

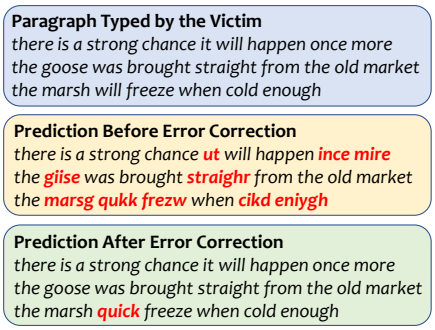
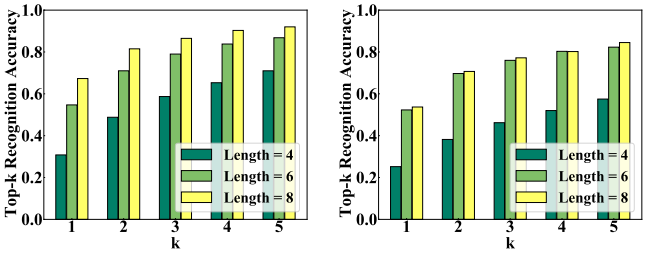


Fig. 16. Examples of Recovered Paragraph.



(a) HTC Vive Pro (b) Oculus Quest

Fig. 17. Performance of Recovering Passwords on Drum-based Typing Without the Enter Key.

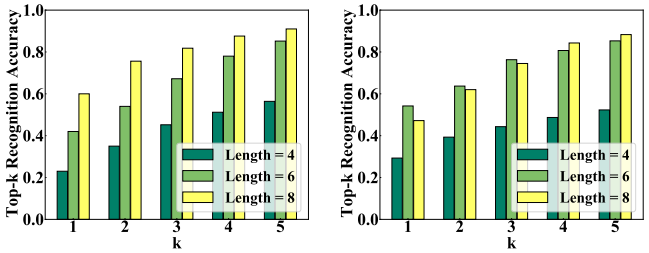
of drum-based typing and laser-based typing for the two VR devices are shown in Figure 17 and Figure 18, respectively. We find that the performance of inferring longer passwords is significantly better than shorter passwords. We believe the reason is that shorter passwords contain less distinguishable positional information, making them more likely to be mixed with false candidates. For instance, the trajectory of the input password "Q-W-E-R" is nearly the same as "W-E-R-T" or "E-R-T-Y", making it hard to distinguish without the "Enter" key, which has a fixed position. However, as longer passwords cover a broader range of areas across the keyboard, their trajectories can obtain a unique pattern more easily. Specifically, under drum-based typing, both headsets achieve over 46% top-3 and over 58% top-5 recognition accuracies for passwords of length 4, and over 76% top-3 and over 82% top-5 recognition accuracies for passwords of length 8. As for laser-based typing, both headsets achieve over 44% top-3 and over 52% top-5 recognition accuracies for passwords of length 4, and over 75% top-3 and over 88% top-5 recognition accuracies for passwords of length 8. The promising results indicate that our attack can be easily generalized to a more challenging scenario in which the "Enter" key is not pressed.

VII. Discussions

In this section, we will discuss the reasons why so many sensors lack permission on VR, the underlying causes of this vulnerability, as well as the limitations of the attack. Additionally, we will also discuss potential approaches to improve the management policies for VR sensors.

A. Why No Permission?

We believe one primary reason that so many sensors on VR are zero-permission is that the community generally believes they are "safe" and is not aware of to what extent they can



(a) HTC Vive Pro (b) Oculus Quest

Fig. 18. Performance of Recovering Passwords on Laser-based Typing Without the Enter Key.

leak the user's privacy, as the security and privacy vulnerability issues on VR systems have not received significant attention and we are one of the early studies in this research line. Additionally, asking for too many permissions from the user for different sensors lowers the usability and may raise too much of a burden when having a great number of sensors for the VR system. Another critical factor is that most users do not pay enough attention to the permission systems and will tend to grant permissions upon request. A usability study on Android demonstrates that only 17% of the users will pay attention to permission warnings during application installation [18], indicating that even if permission exists, a malware can simply ask for permission to sensor data under some pretended reason, and then launch the attack without being noticed.

B. Limitation

One limitation of our attack is that it requires the adversary to possess knowledge of the keyboard layout (e.g., QWERTY), which means it cannot be effective on a keyboard with a randomized layout. To mitigate this vulnerability, developers could randomly change the position of the keys on the keyboard, making them different from the standard QWERTY layout [57], [56], [7], change the location of the keyboard in the VR space after each character is entered (keyboard jitter) [40], alter the shape of the keyboard, making it not a rectangle (keyboard wrapping) [40], or circularly shift keys in each row/column by a random number (row/column shift) [32]. It should be noted that implementing these defense mechanisms may lead to reduced usability and inconvenience for users, as it may take more time and effort to type [43]. Specifically, row/column shift may increase the typing time by 1.5 times [32], and most users may be less willing to use a wrapped keyboard due to unfamiliarity with the layout [40].

C. Refine Sensor Management Policy in VR

Permission-based Sensor Management. One potential approach to improving VR sensor management is to implement a permission-based scheme similar to the one used in Android. Under this approach, users would be required to grant permission for VR apps or webpages to access specific sensors during both installation and runtime.

Privacy-aware Sensor Management. While adding permissions can enhance user privacy to some extent, it is important to note that once permissions are granted, users may

not have full knowledge or control over how their sensor data is used. To increase the transparency and control over data usage, a privacy-aware framework upon the sensor management scheme [55], [39] could be developed. Particularly, the framework can provide information to help users understand the context of sensor data usages, such as sensor types, starting time, sensor usage duration, and the running status of the app that is collecting sensor data (i.e., foreground or background). Based on the contextual information, the framework can allow users to create and update customized access control policies for all the sensors. For instance, this would enable users to restrict sensor data access for background apps when entering sensitive information (e.g., passwords, contact information). Furthermore, the framework can quantify the quality of sensor data supplied to VR apps and allow the user to adjust the data qualities (e.g., sampling rate, resolution) for a specific VR app.

Hardware Refinement. Another potential solution is to integrate indicator lights into VR controllers, which would alert users to any malicious sensor usage in the background. This is especially important for on-board microphones and front-facing cameras, which can record highly sensitive information. As for motion and position sensors, these lights can indicate whether they are being recorded in the background (i.e., additional VR sessions to the primary VR app). By incorporating indicator lights, users can be more aware of the status of their sensors and whether they are being used without their knowledge or consent.

VIII. Related Work

Attacks on VR & Virtual Keyboards. Most of the initial research on exploring the security and privacy implications of VR systems focused on user authentication [28], [21], [20], [33], [30], [12]. A recent work [10] has even introduced a variety of new VR-based attacks that target the user's experience. They demonstrate attacks that can disorient the user, control hardware used for the VR experience, inject images in the user's field of vision, and encourage the user to move to particular locations that may cause them to hit nearby objects. Shi et al. [44] proposed Face-Mic, an eavesdropping attack that leverages motion sensors on VR headsets to infer the user's live speech and identity. As for keystroke detection attacks in VR, a study by Chen et al. [11] showed that VR headsets could be configured with cameras in order to spy on the phone keystrokes of nearby persons. Ling et al. [26] developed side-channel attacks on the Samsung Gear VR system that can be used to infer the passwords entered on the laser-based VR keyboard using the positions and angles of the headset and pointer controller. While this study has shown the feasibility of such attacks, the proposed attacks only targeted a smartphone-based VR system, and the required strong assumption greatly reduces the attack feasibility in practice. Meteriz et al. [34] present a keylogging inference attack to infer user inputs typed with in-air tapping keyboards in Augmented Reality (AR), and Luo et al. [29] propose a similar attack on in-air tapping keyboards in Mixed Reality (MR). However, the typing mechanisms considered in these two studies are hand-gestured-based typing captured by the AR/MR device's front cameras, therefore these two approaches are not applicable to drum- and laser-based typing mechanisms in VR that leverage controllers. Different from AR and MR, controllers have been considered as the major interaction interface in VR compared with hands, as many popular VR headsets (e.g., HTC Vive Pro) and apps (e.g., VRChat, Bigscreen) don't support hand tracking without additional software or hardware accessories.

Different from these studies, our work validates the attack's feasibility on the two mainstream VR systems (i.e., HTC Vive Pro and Oculus Quest), for two interactive methods of typing, under more practical attack models. Additionally, VR-Spy [5] detects keystrokes in the VR space via channel state information (CSI) from WiFi signals, which could be another attack vector in this domain. However, VR-Spy requires labeled data from victim for training and the victim's position is fixed, which largely limits its practicality.

Other Keystroke Attacks. There has been active research on snooping keystrokes on physical keyboards/PIN pads leveraging various side-channels, including audio-based [58], [27], EM-based [50], motion sensor-based [54], [52], and CSI-based [17]. Additionally, many other research have focused on attacking soft keyboards on touch screen for modern smartphones leveraging the built-in motion sensors of the smartphone [38], [35], [9] or the motion sensors on the user's smartwatch [53]. While certain principle concepts of these keystroke attacks remain valid in the developing era of VR devices, we again find ourselves in a new attack space introduced by further technological advancements and more comprehensive proof-of-concept validation. And as VR devices are showing signs of popularity gains like that of smartphones, the underlying threat of the unrestricted VR sensors deserves to have greater attention.

IX. Conclusion

In this paper, we thoroughly examined the trustworthiness of embedded sensors in VR systems and found that their data can be easily accessed by an adversary. We further explore the severity of this privacy leakage in the context of keystroke snooping in VR. The adversary doesn't possess any knowledge about the victim's VR system setting in our considered attack model. Extensive experiments involving two mainstream VR systems and different typing mechanisms demonstrated the effectiveness of our attack on recognizing the victim's keystroke inputs, including both random passwords and natural language text. We hope this study can provide insights into the future design of sensor management policies in VR and help increase the trustworthiness of VR systems.

Acknowledgment

We would like to thank our anonymous reviewers for their insightful feedback. This work was supported in part by NSF grants CNS2114220, CNS2120396, CCF2211163, CNS2114161, CNS2201465, CNS2152669, OAC2139358, and ECCS2132106.

REFERENCES

[1] The best vr apps in 2022. https://www.creativebloq.com/features/best-vr-apps, 2022.

[2] Demo videos of stealthy sensor data collection. https://sites.google.com/view/vr-key-logger, 2022.

[3] Most popular apps in oculus. https://www.oculus.com/experiences/quest/section/1453026811734318/\#/?_k=xlg1bx, 2022.

[4] Most popular apps in vive port. https://www.viveport.com/app.html?product_list_order=popular, 2022.

[5] Abdullah Al Arafat, Zhishan Guo, and Amro Awad. Vr-spy: A side-channel attack on virtual key-logging in vr headsets. In 2021 IEEE Virtual Reality and 3D User Interfaces (VR), pages 564–572. IEEE, 2021.

[6] ARPost. The benefits of virtual reality in banking. https://arpost.co/2020/04/22/benefits-virtual-reality-banking/, 2020.

[7] S Arun Kumar, R Ramya, R Rashika, and R Renu. A survey on graphical authentication system resisting shoulder surfing attack. In Advances in Artificial Intelligence and Data Engineering, pages 761–770. Springer, 2021.

[8] Costas Boletsis and Stian Kongsvik. Controller-based text-input techniques for virtual reality: An empirical comparison. International Journal of Virtual Reality (IJVR), 19(3), 2019.

[9] Liang Cai and Hao Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. In HotSec, 2011.

[10] Peter Casey, Ibrahim Baggili, and Ananya Yarramreddy. Immersive Virtual Reality Attacks and the Human Joystick. IEEE Transactions on Dependable and Secure Computing, 18(2):550–562, 2021.

[11] Song Chen, Zupei Li, Fabrizio Dangelo, Chao Gao, and Xinwen Fu. A Case Study of Security and Privacy Threats from Augmented Reality (AR). In 2018 International Conference on Computing, Networking and Communications (ICNC), pages 442–446. IEEE, 2018.

[12] Yuxin Chen, Zhuolin Yang, Ruben Abbou, Pedro Lopes, Ben Y Zhao, and Haitao Zheng. User authentication via electrical muscle stimulation. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pages 1–15, 2021.

[13] HTC Corporation. Vive port. https://www.viveport.com/, 2021.

[14] HTC Corporation. Vive sync: The future of meetings. https://sync.vive.com/, 2021.

[15] Valve Corporation. Openvr. https://partner.steamgames.com/doc/features/steamvr/openvr, 2021.

[16] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In kdd, volume 96, pages 226–231, 1996.

[17] Song Fang, Ian Markwood, Yao Liu, Shangqing Zhao, Zhuo Lu, and Haojin Zhu. No training hurdles: Fast training-agnostic attacks to infer your typing. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 1747–1760, 2018.

[18] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In Proceedings of the eighth symposium on usable privacy and security, pages 1–14, 2012.

[19] VR Focus. With gear vr gone the samsung xr service is shutting down. https://www.vrfocus.com/2020/05/with-gear-vr-gone-the-samsung-xr-service-is-shutting-down/, 2020.

[20] Markus Funk, Karola Marky, Iori Mizutani, Mareike Kritzler, Simon Mayer, and Florian Michahelles. LookUnlock: Using Spatial-Targets for User-Authentication on HMDs. In Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems, CHI EA '19, pages 1–6. Association for Computing Machinery, 2019.

[21] Ceenu George, M. Khamis, Emanuel von Zezschwitz, Marinus Burger, Henri Schmidt, Florian Alt, and Heinrich Hußmann. Seamless and Secure VR : Adapting and Evaluating Established Authentication Systems for Virtual Reality. In Network and Distributed System Security Symposium, NDSS '17, 2017.

[22] Google. Daydream labs: exploring and sharing vr's possibilities. https://developers.googleblog.com/2016/05/daydream-labs-exploring-and-sharing-vrs.html, 2016.

[23] VRChat Inc. Vrchat. https://hello.vrchat.com/, 2021.

[24] Jasoren. Vr military training – the next step of combat evolution. https://jasoren.com/vr-military-training-the-next-step-of-combat-evolution/, 2018.

[25] W Kündig. A least square fit program. Nuclear Instruments and methods, 75(2):336–340, 1969.

[26] Zhen Ling, Zupei Li, Chen Chen, Junzhou Luo, Wei Yu, and Xinwen Fu. I know what you enter on gear vr. In 2019 IEEE Conference on Communications and Network Security (CNS), pages 241–249. IEEE, 2019.

[27] Jian Liu, Yan Wang, Gorkem Kar, Yingying Chen, Jie Yang, and Marco Gruteser. Snooping keystrokes with mm-level audio ranging on a single phone. In Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, pages 142–154, 2015.

[28] Yujun Lu, BoYu Gao, Jinyi Long, and Jian Weng. Hand motion with eyes-free interaction for authentication in virtual reality. In 2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), pages 714–715. IEEE, 2020.

[29] Shiqing Luo, Xinyu Hu, and Zhisheng Yan. Holologgger: Keystroke inference on mixed reality head mounted displays.

[30] Shiqing Luo, Anh Nguyen, Chen Song, Feng Lin, Wenyao Xu, and Zhisheng Yan. OcuLock: Exploring Human Visual System for Authentication in Virtual Reality Head-mounted Display. In NDSS, 2020.

[31] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[32] Anindya Maiti, Murtuza Jadliwala, and Chase Weber. Preventing shoulder surfing using randomized augmented reality keyboards. In 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pages 630–635. IEEE, 2017.

[33] Florian Mathis, Hassan Ismail Fawaz, and Mohamed Khamis. Knowledge-driven Biometric Authentication in Virtual Reality. In Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, CHI EA '20, pages 1–10. Association for Computing Machinery, 2020.

[34] Ulkü Meteriz-Yıldıran, Necip Fazıl Yıldıran, Amro Awad, and David Mohaisen. A keylogging inference attack on air-tapping keyboards in virtual environments.

[35] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. Tapprints: Your finger taps have fingerprints. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys '12, page 323–336. Association for Computing Machinery, 2012.

[36] Mozilla. Firefox reality. https://mixedreality.mozilla.org/firefox-reality/, 2021.

[37] MIT News. Bringing the benefits of in-person collaboration to the virtual world. https://news.mit.edu/2020/spatial-vr-collaboration-0710, 2020.

[38] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Ying Zhang. Accessory: password inference using accelerometers on smartphones. In HotMobile '12, 2012.

[39] Nisarg Raval, Ali Razeen, Ashwin Machanavajjhala, Landon P Cox, and Andrew Warfield. Permissions plugins as android apps. In Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services, pages 180–192, 2019.

[40] Eric Tanner Reed. The Use of Alternative Keyboard Structures to Prevent Shoulder Surfing Attacks in Augmented Reality. PhD thesis, Christopher Newport University, 2020.

[41] Grand View Research. Virtual reality market size, share & trends analysis report by technology (semi & fully immersive, non-immersive), by device (hmd, gtd), by component (hardware, software), by application, and segment forecasts, 2021 - 2028. https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-market\#, 2021.

[42] EH Rothauser. Ieee recommended practice for speech quality measurements. IEEE Trans. on Audio and Electroacoustics, 17:225–246, 1969.

[43] Young Sam Ryu, Do Hyong Koh, Brad L Aday, Xavier A Gutierrez, and John D Platt. Usability evaluation of randomized keypad. Journal of Usability Studies, 5(2):65–75, 2010.

[44] Cong Shi, Xiangyu Xu, Tianfang Zhang, Payton Walker, Yi Wu, Jian Liu, Nitesh Saxena, Yingying Chen, and Jiadi Yu. Face-mic: inferring live speech and speaker identity via subtle facial dynamics captured by ar/vr motion sensors. In Proceedings of the 27th Annual International Conference on Mobile Computing and Networking, pages 478–490, 2021.

[45] Facebook Technologies. Oculus platform sdk. https://developer.oculus.com/downloads/package/oculus-platform-sdk, 2021.

[46] Facebook Technologies. Supplemental oculus data policy. https://www.oculus.com/legal/privacy-policy/, 2021.

[47] Tvori. Tvori. https://tvori.co/, 2021.

[48] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. Nature methods, 17(3):261–272, 2020.

[49] Immersion VR. Vr for tourism. https://immersionvr.co.uk/about-360vr/vr-for-tourism/, 2020.

[50] Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In USENIX Security Symposium, 2009.

[51] W3C. Webxr device api. https://immersive-web.github.io/webxr/, 2021.

[52] Chen Wang, Xiaonan Guo, Yan Wang, Yingying Chen, and Bo Liu. Friend or foe? your wearable devices reveal your personal pin. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, pages 189–200, 2016.

[53] Chen Wang, Jian Liu, Xiaonan Guo, Yan Wang, and Yingying Chen. Wristspy: Snooping passcodes in mobile payment using wrist-worn wearables. In IEEE International Conference on Communications, 2019.

[54] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. Mole: Motion leaks through smartwatch sensors. In Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, pages 155–166, 2015.

[55] Zhi Xu and Sencun Zhu. Semadroid: A privacy-aware sensor management framework for smartphones. In Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, pages 61–72, 2015.

[56] Dhruv Kumar Yadav, Beatrice Ionascu, Sai Vamsi Krishna Ongole, Aditi Roy, and Nasir Memon. Design and analysis of shoulder surfing resistant pin based authentication mechanisms on google glass. In International conference on financial cryptography and data security, pages 281–297. Springer, 2015.

[57] Ruide Zhang, Ning Zhang, Changlai Du, Wenjing Lou, Y Thomas Hou, and Yuichi Kawamoto. Augauth: Shoulder-surfing resistant authentication for augmented reality. In 2017 IEEE International Conference on Communications (ICC), pages 1–6. IEEE, 2017.

[58] Li Zhuang, Feng Zhou, and J. Doug Tygar. Keyboard acoustic emanations revisited. ACM Transactions on Information and System Security, 13:3, 2009.

APPENDIX A
EXTRACTING SENSOR DATA USING ANDROID SDK

To demonstrate that the sensor data of victim's controllers can be logged in the back-end on Android systems, we also implement a VR application leveraging Android SDK APIs to evaluate our proposed eavesdropping attack. For most VR systems, they are built grounded on Android systems, which provide open interfaces for VR application developers to achieve their proposed functions. Specifically, for motion and position sensors, which are denoted to be `TYPE_(SENSORS_NAME)` in Android development APIs, can be monitored via the creation of `SensorEvent` created by Android systems. For each `SensorEvent`, they continuously track the multi-dimensional arrays (e.g., acceleration and position of the x, y, z coordinates) of sensor values. Meanwhile, the clicking event will be detected through `MotionEvent`. Once the user clicks the button on the VR controllers, the attribute `onTouchEvent()` of `MotionEvent` will be activated. To sum up, the comprehensive support of Android SDK APIs makes it possible to track user's motion and clicking events through an Android application, which can be leveraged by our attack in VR systems.

APPENDIX B
TYPING MECHANISMS & KEYBOARD LAYOUTS IN POPULAR VR APPLICATIONS

We summarize the typing mechanism & keyboard layouts in popular VR applications in Table II.

APPENDIX C
SELECTED HARVARD SENTENCES

The randomly selected Harvard sentences are listed in Table III.

APPENDIX D
PERFORMANCE EVALUATION OF MACHINE LEARNING ALGORITHMS

We further examine the possibility of using state-of-the-art machine learning algorithms on the sensor data for keystroke inference. Specifically, the adversary slices a window of 0.5 seconds of sensor data centered at each detected keystroke. The adversary then extracts 13 time-domain features from the window for each dimension, including minimum, maximum, median, variance, std, abs-mean, cv, skewness, kurtosis, first quartiles, second quartiles, third quartiles, inter quartile-range. Given the 3D position and orientation sensor data, each keystroke forms a 78-dim feature vector. We then utilize four types of machine learning classifiers: Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and a two-dense-layer Deep Neural Network (DNN) with 50 neurons in each layer. In addition to time-domain features, we also apply Recurrent Neural Network (RNN) on the raw time-series sensor data to measure the temporal dependencies and extract high-level features. Specifically, We utilize a 2-layer RNN architecture, and the extracted features are fed into a dense layer for classification. We use the adversary's data as the training data and other participants' data as testing data, therefore the training/testing ratio is 1 : 6 for each typing mechanism. We find that the average single keystroke recognition accuracies of the five machine-learning-based approaches for four typing mechanisms are only 5.96%, 12.33%, 8.59%, and 12.99%. As the position and orientation of the virtual keyboard differ a lot between the adversary and the victim, their sensor data exhibits completely different patterns, making it hard to use machine-learning-based approaches to recognize the victim's typed inputs.

APPENDIX E
PERFORMANCE EVALUATION ON ANDROID APP

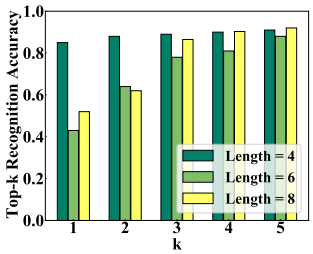
The developed malicious Android app is installed on the Oculus Quest. We use the default virtual keyboard of the Android Notes app for both drum-based and laser-based typing. We find that keyboard layouts in Android apps are very similar to the Oculus default keyboard, therefore we directly use the previously reconstructed keyboard using Oculus as the adversary's reference pattern. We recruit one participant to act as victim and perform the same data collection procedure as aforementioned in Section VI-A. Figure 19 (a) and (b) illustrate the performance of the password inference

TABLE II
TYPING MECHANISMS & KEYBOARD LAYOUTS IN POPULAR VR APPLICATIONS.

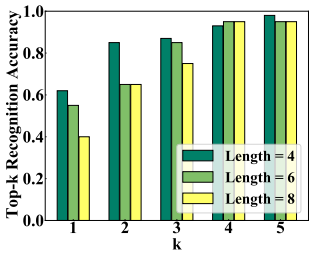
Application	Usage	Typing Mechnism	Keyboard Layout
Text Editors in Oculus Quest	Typing	Drum/Laser	QWERTY, numeric on the top, Enter on the right
Steam VR	Social Platform & Shopping	Laser	QWERTY, numeric on the top, Enter on the right
Vive Port	Shopping	Drum	QWERTY, numeric on the top, Enter on the right
Vive Video	Video Player	Drum	QWERTY, numeric on the top, Enter on the right
Youtube VR	Video Player	Laser	QWERTY, numeric on the left, Enter on the right
Vive Sync	Online Meeting & Collaboration	Laser	QWERTY, numeric on the top, Enter on the right
Tvori	Design/Creation	Drum	QWERTY, numeric on the top, Enter on the right
Google Daydream Lab	Design/Creation	Drum	QWERTY, no numeric & Enter
Gravity Sketch VR	Design/Creation	Drum	QWERTY, numeric on the top, no Enter
VRChat	Social Platform	Laser	QWERTY, numeric on the top, Enter on the bottom
Alterspace VR	Social Platform	Laser	QWERTY, numeric on the top, Enter on the bottom
Bigscreen	Social Platform	Laser	QWERTY, numeric on the top, Enter on the bottom
Rec Room	Social Platform	Laser	QWERTY, numeric on the top, Enter on the right

TABLE III
SELECTED HARVARD SENTENCES

Index	Sentence
1	The fruit of a fig tree is apple shaped
2	Hold the hammer near the end to drive the nail
3	There is a strong chance it will happen once more
4	The goose was brought straight from the old market
5	The case was puzzling to the old and wise
6	The pup jerked the leash as he saw a feline shape
7	The weight of the package was seen on the high scale
8	A good book informs of what we ought to know
9	The marsh will freeze when cold enough
10	The steady drip is worse than a drenching rain



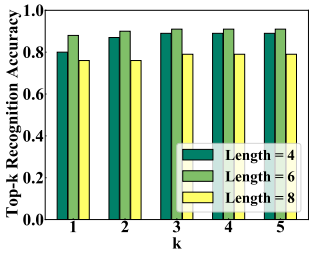
(a) Laser-based Typing



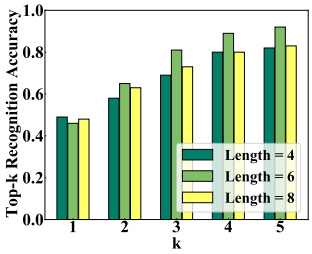
(b) Drum-based Typing

Fig. 19. Performance of Recovering Passwords on Android.

attack on the Android app for laser-based and drum-based typing, respectively. We achieve (85%, 43%, and 52%) top-1, (89%, 78%, 87%) top-3, and (91%, 88%, and 92%) top-5 accuracies on inferring passwords with length 4, 6, and 8 for laser based typing, and (62%, 55%, and 40%) top-1, (87%, 85%, 75%) top-3, and (98%, 95%, and 95%) top-5 accuracies for drum-based typing. Additionally, we achieve 94.8% and 91.6% WRR for the paragraph inference attack for laser- and drum-based typing, respectively. These promising results demonstrate that our attack can be easily generalized to Android VR systems.



(a) HTC Vive Pro



(b) Oculus Quest

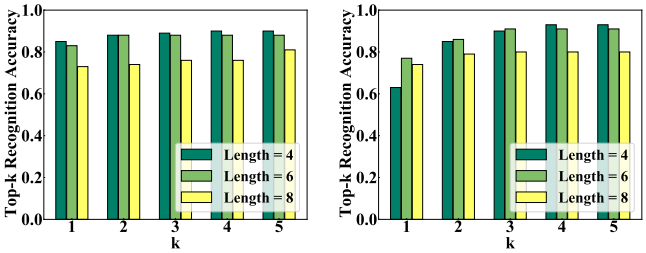
Fig. 20. Impact of Upper Cases and Symbols on Drum-based Typing.

APPENDIX F
IMPACT OF UPPER CASES AND SYMBOLS

We further extend the password recovery attack by involving upper cases and symbols. For each typing scenario, one participant acts as the adversary, and the other participant is treated as the victim. In addition to the aforementioned 38 keys, we also include 10 different symbols (i.e., - = [ ] \ ; ' , . /) and the CapsLock key in the data collection session. For each victim, we generate 10 different passwords following the aforementioned methods with lengths of 4, 6, and 8. We ensure there is at least one upper case and one symbol in each password. The results of drum-based typing and laser-based typing for the two VR devices are shown in Figure 20 and Figure 21, respectively. We find that the impact of using upper cases and symbols in the password on the attack performance is very subtle: under drum-based typing, both headsets achieve over 69% top-3 and over 79% top-5 recognition accuracies for passwords of all three lengths. Laser-based typing performs even better: both headsets have over 75% top-3 and over 81% top-5 recognition accuracies. The promising results indicate that our attack can be easily generalized to more complex passwords.

APPENDIX G
IMPACT OF DIFFERENT KEYBOARD LAYOUTS

We further investigate the worst-case scenario, in which the adversary does not possess any prior knowledge of the victim's keyboard layout and uses a completely different keyboard to infer the victim's typed paragraph. Since all virtual keyboards in Table II follow the standard QWERTY layout for alphabets,



(a) HTC Vive Pro (b) Oculus Quest

Fig. 21. Impact of Upper Cases and Symbols on Laser-based Typing.

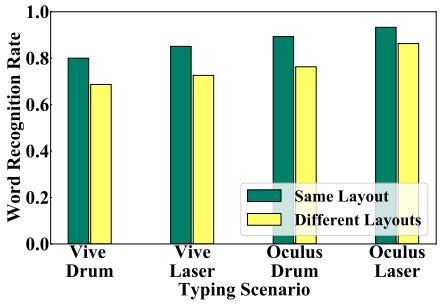


Fig. 22. Impact of different keyboard layouts

we believe our paragraph inference attack would still be effective even if the adversary's VR keyboard is different from the one used by the victim. To validate this, for each typing scenario (i.e., drum- and laser-based typing on Vive and Oculus, in total four different scenarios), we let the adversary use the three different keyboards utilized in other typing scenarios to infer the paragraph typed by the victims. The average WRR for each scenario is shown in Figure 22. Although the attack performance decreases due to different structures and distances between keys, we can still achieve an average WRR of 68.7% and 72.6% for drum-/laser-based typing on Vive, and 76.3% and 86.4% WRR for drum-/laser-based typing on Oculus. These promising results show the potential generality of our attack across different keyboard layouts.

APPENDIX H
OTHER KEYBOARD SCHEMES

We mainly focus on the QWERTY keyboards in this paper as this layout is utilized in almost every mainstream VR application. The adversary can include the keyboards of other layouts (e.g., QWERTZ, AZERTY, and Dvorak) during the keyboard reconstruction phase to improve the generalizability of the attack.