Going through the motions: AR/VR keylogging from user head motions

Carter Slocum, Yicheng Zhang, Nael Abu-Ghazaleh and Jiasi Chen
University of California, Riverside

Abstract

Augmented Reality/Virtual Reality (AR/VR) are the next step in the evolution of ubiquitous computing after personal computers to mobile devices. Applications of AR/VR continue to grow, including education and virtual workspaces, increasing opportunities for users to enter private text, such as passwords or sensitive corporate information. In this work, we show that there is a serious security risk of typed text in the foreground being inferred by a background application, without requiring any special permissions. The key insight is that a user's head moves in subtle ways as she types on a virtual keyboard, and these motion signals are sufficient for inferring the text that a user types. We develop a system, TyPose, that extracts these signals and automatically infers words or characters that a victim is typing. Once the sensor signals are collected, TyPose uses machine learning to segment the motion signals in time to determine word/character boundaries, and also perform inference on the words/characters themselves. Our experimental evaluation on commercial AR/VR headsets demonstrate the feasibility of this attack, both in situations where multiple users' data is used for training (82% top-5 word classification accuracy) or when the attack is personalized to a particular victim (92% top-5 word classification accuracy). We also show that first-line defenses of reducing the sampling rate or precision of head tracking data are ineffective, suggesting that more sophisticated mitigations are needed.

1 Introduction

While Augmented and Virtual Reality (AR/VR) headsets have existed for many years [54], they have not been widely available to consumers until recently [21]. There has been widespread and growing adoption commercially with an estimated 26 million devices already in circulation [10]. Applications of AR/VR continue to grow beyond entertainment, to areas such as education, training, social media, and remote work [32, 33]. Such applications may require the entry of sensitive data in the form of text, entered by interacting with a virtual keyboard rendered in the headset. Private messages, passwords, and PIN codes might be entered in this manner. At the same time, AR/VR platforms now provide the ability to run multiple applications simultaneously [34, 35]. Examples include multiple virtual web browser windows open and visible at the same time, or a chatting application overlaid on top of a virtual game.

This work studies the security risks posed by the confluence of these two trends – AR/VR users typing sensitive information in virtual spaces, and support for multi-app scenarios. In particular, we find that without any special permissions, background apps can infer the typed words and characters of a foreground app. The key insight is that a user's head moves subtly as she types on a virtual keyboard, and these head motions are enough to accurately infer what she is typing. Moreover, this attack is feasible because the sensor data about these head motions are freely available to a malicious background app. Fundamentally, this is because all applications, even those running in the background, need to continuously estimate the user's head pose in order to update their rendering in response to a user's head motions.

Recent works have shown that it is possible to infer the sensitive typed information through side-channels, but only through the same modality (*e.g.,* hand or head motions). For example, Meteriz-Yildiran et al. [36] show hand tracking data can be used to reconstruct hand-typed characters. Holologger [29] shows head tracking data can be used to reconstruct characters typed using "head gaze commit," where the user points her head at the desired keys. In contrast, we show that *head* tracking alone is sufficient to steal *hand*-entered text. This was initially surprising to us because qualitatively, we did not observe much head motions when AR/VR users were typing text.

To infer hand-entered text accurately from head motions alone, we faced several challenges. (1) There is an unclear relationship between a user's head motions and what they are typing. The idea is that some character sequences could result in larger head motions (*e.g.,* "a" followed by "p"), while

other sequences could result in smaller head motions (*e.g.,* "a" followed by "s"), helping differentiate between the two cases. (2) From the head motions alone, it is unclear when the user is actively typing characters or words. The idea is that text entry applications force the user to make larger movements of the head corresponding to the start and end of the text entry, and certain key presses (such as the space bar) are strong indicators of a break between words. (3) How to mitigate such attacks, without degrading user experience, is unclear. The issue is that blocking an app's access to headset tracking data would cause the app's rendering to freeze as the user moves around, breaking the immersion with the virtual world [24].

We call our system TyPose, since we estimate hand *typed* characters/words from the victim's head *pose*. Overall, we make the following contributions:

- To the best of our knowledge, TyPose is the first attack that infers the private hand-entered text of an AR/VR user using only head motion tracking information, requiring zero permissions by a malicious background app (Section 2).
- We design machine learning techniques to automatically infer words and characters typed by a user, including a Segmenter to divide a stream of sensor readings into the corresponding words/characters and a Classifier to infer the text corresponding to those segments (Section 3).
- We collect traces of AR/VR typing behavior from 21 users and evaluate our attack on these traces. The results show that TyPose can detect segments and identify words with high accuracy; for example, a top-5 classification accuracy of 82% for inferring words (Section 4). We also explore the feasibility of an end-to-end attack (Section 5).
- We evaluate first-line mitigation strategies, including downsampling the head tracking sensor stream or reducing their floating point precision. We find that the proposed attacks are generally robust to such mitigations, and thus these sensor streams may need to be further blocked to potentially malicious background apps (Section 6).

2 Background and Threat Model

In this section, we first introduce the relevant background with respect to AR/VR headsets (Section 2.1). We then define the threat model (Section 2.2), and discuss some of the intuitions and challenges behind the proposed attack (Section 2.3).

2.1 Background on VR

Head motion tracking. Standalone AR/VR headsets track their position and orientation in the real world using a combination of camera and inertial measurement unit (IMU) sensor readings. This allows the AR/VR platform to render believable and immersive scenes, updating the display as the user moves. Without headset tracking, the scene would appear "frozen in place" even as the user moved her head. Laggy or inaccurate headset tracking are key causes of motion sickness, particularly in VR [24]. Thus, headset tracking is a fundamental aspect of AR/VR, and it is standard practice to allow access to head tracking data to all applications to make sure they can continue rendering.

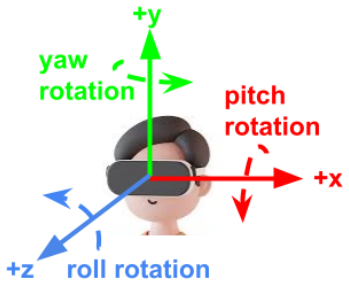


Figure 1: Illustration of the x, y, z axes on the VR headset. The accelerometer measures the linear acceleration along these axes, and gyroscope measures the angular velocity around them.

The Intertial Measurement Unit (IMU) in an AR/VR headset contains an accelerometer and gyroscope. The IMU tracks 6 Degrees of Freedom (DoF) as shown in Fig. 1: 3 DoF correspond to linear acceleration along the x, y, z axes, measured by the accelerometer in m/s^2, and 3 DoF correspond to angular velocity along the x, y, z axes, measured by the gyroscope in rad/s. These raw readings from the IMU are passed to the AR/VR software so it can update the display appropriately. The raw readings are integrated to obtain the position and orientation of the headset (also known as "pose").

Multiple AR/VR applications. Recent improvements to AR/VR platforms allow multiple applications to run simultaneously. These apps can run simultaneously in the foreground [34], or some apps can be suspended (moved to the background) in order for other apps to run (in the foreground) [35]. An example of the former is opening up a virtual web browser and a TV show app simultaneously. An example of the latter shown in Fig. 2, where the user pauses her virtual drumming game ② to message a friend ④ using a social media service that may require a PIN or password to be entered in order to log in. While the background app is considered paused, it is not completely suspended in its execution. Users are able to interact with the foreground application while the view of the background app continues to update if the user moves her head, and animations continue to play. In other words, the background application still receives 6 DoF headset tracking data in order to render correctly and preserve the immersion of the user.

2.2 Threat Model

Attack overview. Our threat model assumes that the user has installed a VR application with malicious code. The at-

Figure 2: Attack scenario. (1) The foreground app (Facebook Messenger) displays on top of the (2) background app (Beat Saber), which continues to render using real-time head motion tracking. (3) The victim types messages using the system keyboard into the (4) text entry field. (5) The controller inputs are not available to background app (2) while the foreground app is open.

tack proceeds as follows. The user switches to a new foreground application ①, suspends the malicious application to the background ②, and begins entering sensitive text ④ in the foreground application (for example, a password or work emails) using the system keyboard ③ and VR controller ⑤. The malicious application receives a signal from the VR platform that it is not in focus and logs all headset tracking data. Specifically, the malicious app logs the 6 DoF accelerometer and gyroscope data every frame for later transmission to, or pickup by, the attacker. The attacker's goal is to reconstruct portions of the sensitive text with a reasonable degree of accuracy.

Assumptions. While a malicious app could be installed through physical access to the device by the attacker and then handed to the victim, this assumption is not necessary. The malicious app could simply be developed as a benign app or game with hidden malicious code, and be released through the VR app store (*e.g.,* the Oculus Quest Store). The user would then install the malicious application themselves unknowingly. To retrieve the data from the malicious app, they could implement benign network functionality (*e.g.,* an online leader-board) in order to obtain network access permissions from the user, then abuse those permissions to send the headset tracking logs to a remote attacker [66, 67]. Using

	Train with all users' data	**Train with one victim's data**
Infer words	Scenario 1A	Scenario 1B
Infer characters	Scenario 2A	Scenario 2B

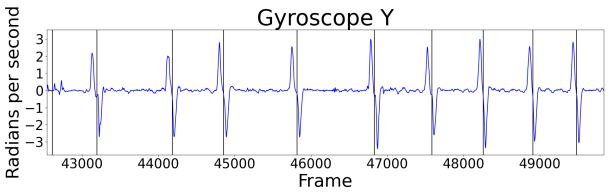
Table 1: Attack taxonomy. TyPose can infer words or characters, and can be trained with multiple users' data or just a single victim's data. For scenario A, the victims' data is excluded from training.

the network in this manner is unlikely to raise concerns from the user, as prior studies have shown that over 70% of Oculus VR app dataflows were not properly disclosed by the privacy policy [55].

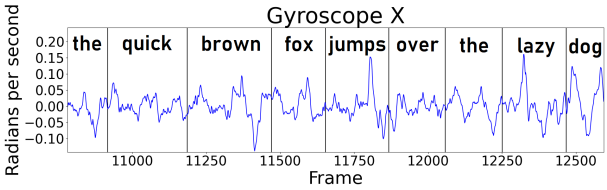
Our threat model assumes that the default VR operating system (in our case, the Meta Quest 2) is active and un-altered, and developer options and privileges are disabled. The sensitive text entered by the user is in the foreground app and therefore not available directly to the background app, nor is the headset or controller position/orientation available (access to this is disabled once the foreground app launches). Headset tracking is enabled to background applications without special permissions. We experimentally verified this by running a real foreground application (Facebook Messenger) and recording the headset tracking data in a custom background app with standard permissions. The above settings are the default in the Meta Quest 2. The attacker does not need access to other sensors such as eye tracking or cameras, network diagnostics, or performance counters. The attacker also does not need information about the system keyboard that the user is using, such as its position, orientation, or size, or the timing of key presses.

Attack taxonomy. We further divide the general attack described above into multiple scenarios, as summarized in Table 1. The attacker may be interested in inferring words typed by the victim (Scenario) 1) or individual characters (Scenario 2). Word inference is useful if the victim is typing full English sentences, while character inference is useful if the victim is typing in a random character sequence (*e.g.,* in a password). We also consider how much ground truth data the attacker has access to. In Scenario type A, the attacker has ground truth data from all users (excluding the victims), and attempts to infer the typing of a particular victim. In Scenario type B, the attacker only has ground truth data from a particular victim, and attempts to infer further typing by the victim.

The ground truth data could be collected from the attacker(s) themselves or from by willing experiment participants. Ground truth data from the victim could be collected by for example, a phishing attempt where the attacker sends chat messages to a victim. As the victim responds, the background app records the victim's head movements, so the attacker has

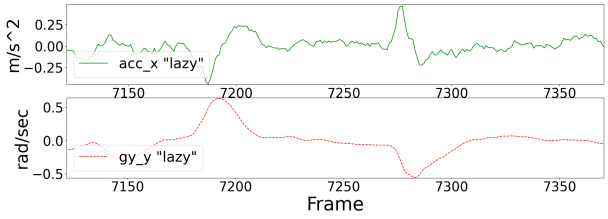


(a) Segmenting sentences is easier when the user must press a distant "submit" button in between words, resulting in large head rotations and spikes in the plot.

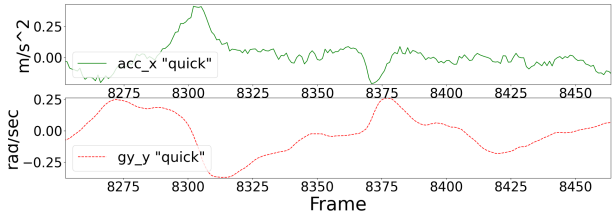


(b) Segmenting sentences is harder when the user only presses the space bar between words, as the word boundaries are not visually distinct.

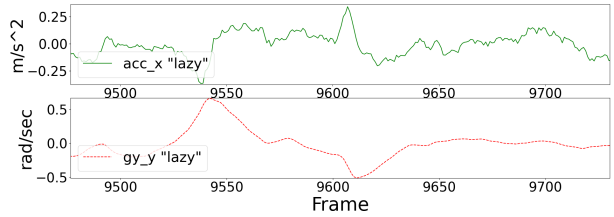
Figure 3: Examples traces of a user's head rotation when typing sentences. The vertical black lines are the ground truth word boundaries. The goal is to segment the sentence into words or characters.



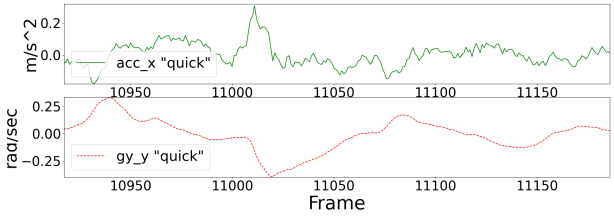
(a) Victim types the word "lazy"



(b) Victim types the word "quick"



(c) Victim types the word "lazy"



(d) Victim types the word "quick"

Figure 4: Example of a user's head rotation and linear acceleration when typing the words "lazy" and "quick", twice each. The traces are quite dissimilar across words, and somewhat dissimilar across trials of the same word.

both the victim's head movements and the ground truth text to train the models. Successful attack samples can also be used to expand the ground truth dataset.

2.3 Illustration of Challenges and Intuition

In this section, we present several motivating examples to illustrate the intuition as well as the challenges in inferring user typing from headset tracking data.

Segmenting sentences into words or characters. A first challenge is to infer when the victim is actively typing when the keyboard is open. Only if we know *when* a user is typing can we then begin to infer *what* is being typed. In certain special cases, finding when the user is typing can be relatively easy, such as when a user types words into a search bar and then has to make a large head movement to press the "search" button. These large head movements provide a strong signal that a word has been entered. In Fig. 3a, we show an example trace of such a case, where the user types a sequence of words and presses a button after every word. The button is located approximately 60° horizontally from the text entry field. This causes distinct spikes in the yaw angular velocity between every word, as shown in Fig. 3a (the black line is the ground truth word boundary). In this situation, an attacker could simply "eyeball" the raw gyroscope data to find word boundaries, or use a simple threshold policy.

The general case is when words boundaries are marked by presses of the space-bar. This is a harder problem because the space-bar is much closer to the other keys being typed and consequently results in far less distinct patterns in the head tracking data. An example trace of a user typing a sentence is shown in Fig. 3b, in terms of the pitch (*i.e.,* headset rotation up and down). Eye-balling the data proves unsatisfactory to find these space-bar-marked word boundaries. However, there are generally large changes in the pitch near the word boundaries, since the user looks down slightly to press the space bar. This suggests that perhaps some patterns can be learned, and motivates our learning-based approach (Section 3.2).

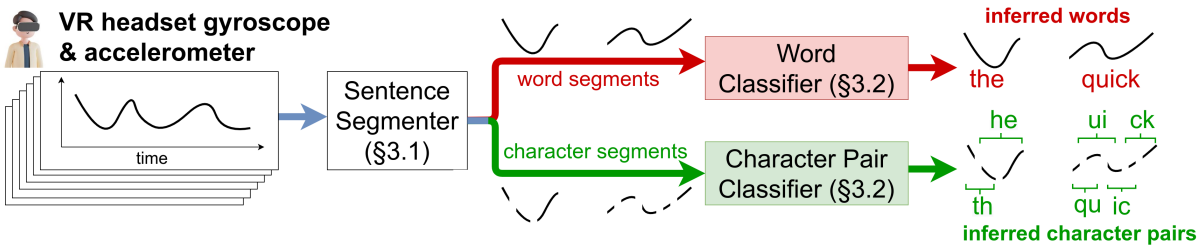


Figure 5: System overview. TyPose takes the 6 DoF VR headset gyroscope and accelerometer sensor readings as input, segments the time series into words (characters), and classifies the words (character pairs).

Inferring what word or characters are typed. Even if the word/character segments are given, TyPose still needs to determine what words/characters are being typed. In Fig. 4, we show an example of a victim typing the same word ("lazy" and "quick"). It can be seen that the traces contain similarities as well as dissimilarities across trials. We were surprised because the head movements of the user were barely discernible visually during the experiment, but the gyroscope and accelerometer were able to pick up enough signal to differentiate the various cases. These experiments provide motivation that TyPose can successfully infer VR user typing (Section 3.3).

3 TyPose's Design

3.1 System Overview

TyPose consists of the following two main modules, as summarized in Fig. 5:

- **Sentence Segmenter:** TyPose determines when the user is actively entering text on the keyboard, as opposed to pausing in between words or in between key entries. We adopt a machine learning approach in order to segment a sentence into words (Scenario 1) or characters (Scenario 2), based solely on the gyroscope and accelerometer readings. TyPose trains two convolutional neural networks (CNNs) for these purposes, respectively.
- **Word Classifier:** The output of the Segmenter is the word boundaries, which are converted into time series segments representing probable words. The Classifier analyzes these segments to infer the typed words. We also experiment with classifying character pairs, given character boundaries. A separate CNN is trained for each task.

In the subsequent sections, we explain the details of each of these modules.

3.2 Sentence Segmenter

Overview. The first problem, since the attacker only has access to the head pose and not key press timings, is to determine when the victim is starting and ending a typed word (Scenario 1), or a typed pair of characters (Scenario 2). In other words, we need to segment a sentence into words, or a sentence into characters. Our first attempt to do this was using conventional auto-segmentation techniques from the time series literature, which unfortunately proved inadequate. These techniques start with a small segment of the data, find a line of best fit, and then grow the segment until the mean squared error from the line passes a threshold [20]. Intuitively, these methods suffer from the inability to recognize common motifs in the time series, and instead seek to segment a time series at points where a nebulous "state change" occurs. While useful for certain tasks, these methods do not leverage the fact that in our application domain, there is a clearly defined motif (*e.g.,* space bar presses in Scenario 1) that is correlated with the start/end of a segment.

TyPose leverages this intuition that there are specific motifs in the data, and that segments occur where the motifs are found. The main issue is that we do not know what exactly these motifs look like in the time series data. Instead, these motifs need to be learned. To solve this, we treat the segmentation problem as a binary classification problem: given a short window of the time series, is it a boundary of a segment? In this way, we transform the problem of finding segments to one of finding boundaries. For sentence segmentation into individual words, the boundaries are the presses of the space key. For sentence segmentation into individual characters, the boundaries are any character press, including spaces.

Beyond accuracy (as we will show later in Section 4), the classifier-based approach has several advantages: there is no need to know the exact dimensions of the keyboard, its position in the victim's field-of-view, or whether the keyboard moves during the typing process. This is because the model is trained on the raw accelerometer and gyroscope data and does not need semantic information about the keyboard or VR scene. For example, even if the keyboard drifts to follow the user's head orientation (as it does in our experiments with the Meta Quest 2), our classifier can still perform well. We believe this is because the sensors measure change, and hence the readings for a given head movement are similar no matter the keyboard's absolute position/orientation.

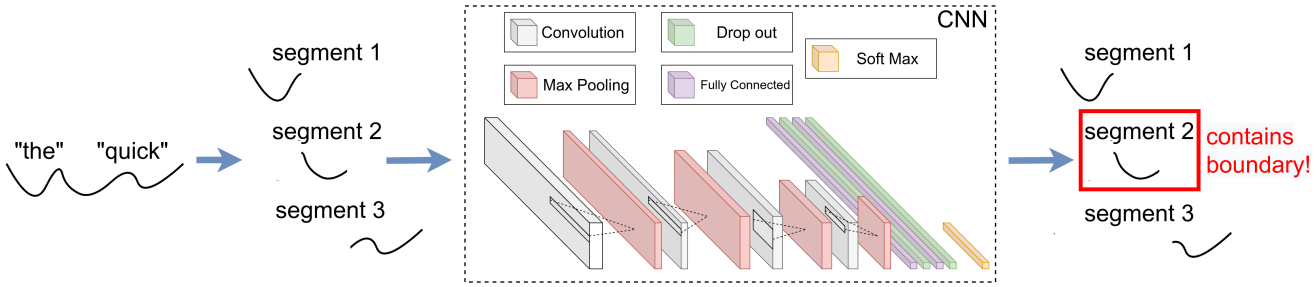


Figure 6: Sentence segmenter design. The example shows finding the word boundaries in a sentence. The same design is used to find character boundaries in a word.

Model design. At a high level, the segmenter takes windowed samples from the 6 DoF time series as input and outputs the probability that a sample contains a boundary or not. From this, TyPose considers the sample as potentially containing a boundary if the probability is above a threshold T. This is illustrated in Fig. 6. The length of the window is parametrized by W.

Our specific classification method is based on CNNs, which have excellent predictive ability in image classification and time series classification problems [45, 65]. Later in Section 4, we also compare against classical k-nearest neighbors (KNN) and Random Convolutional Kernel (ROCKET) time series methods. Essentially, TyPose treats the 6DoF time series window as a 2D "image" of size $6 \times W$, where the value of each "pixel" in the input "image" is the floating point linear acceleration or angular velocity of the headset at each time. The specific model architecture comprises 4 convolutional layers: four 1D kernels with kernel size 3 and 32, 64, 128, 256 units respectively. Finally, the data is fed into two fully connected layers and a soft-max layer. The output dimension of the last fully connected layer is equal to the size of the number of classes (*i.e.,* 2 classes – space or not – in the sentence segmenter, or 2 classes –any character press or not– in the word segmenter). After every convolutional layer, there is a 1D max pooling layer of size 4, and in between each fully connected layer, there is a drop out layer with a chance of 0.5 in order to reduce over-fitting. The CNN uses ReLU activation functions and the sparse categorical cross entropy loss function.

The intuition behind this CNN architecture was to initially convolve the data points from the gyroscope and accelerometer individually along the time axis, rather than jointly convolving gyroscope and accelerometer readings together. In other words, we used 1D kernels in the early layers so that the matrix entries corresponding to gyroscope or accelerometer readings were mostly kept separate. We found empirically that such separation resulted in better performance. Otherwise, the model architecture was close to those found in previous literature [19, 27, 43].

3.3 Word Classifier

Overview. Given the word segments, (either the ground truth or those predicted by the Segmenter from Section 3.2, the next problem is to determine what words are being typed. To the best of our knowledge, no models for predicting typed VR characters based solely on head pose exist. We model the attacker's problem as a classification problem, with the head pose 6 DoF time series data as input, and the typed text as the output classes. In Scenario 1 for word inference, the output classes are the words being typed. We also experiment with Scenario 2 for character inference, where the output classes are the character pairs being typed.

Model design. The word or character pair classifier use a similar CNN model to the segmenter (Section 3.2), with the main difference being the size of the input and the number of possible output classes. Since the word/character segments output from the Segmenter can be of variable length, depending on the user, TyPose considers the maximum length word/character, pads shorter samples with 0's, and inputs them to the word/character pair classifier. Changing the CNN design to predict from a different number of possible output classes simply requires changing the output size of the final fully connected layer. In Scenario 1, the number of output classes is equal to the size of the word corpus being trained on. In Scenario 2, the number of output classes is equal to the number of unique character pairs under consideration. In this work, we consider common words and character pairs in the English language (details in Section 4).

4 Evaluation

4.1 Data Collection

Data collection application. In order to show the viability of an end-to-end attack on current AR/VR systems, we created a malicious background VR application to log the headset's accelerometer and gyroscope readings. The application was created in Unity version 2020.3.26f1 [57] and deployed on the Meta Quest 2. A screenshot of the application is shown

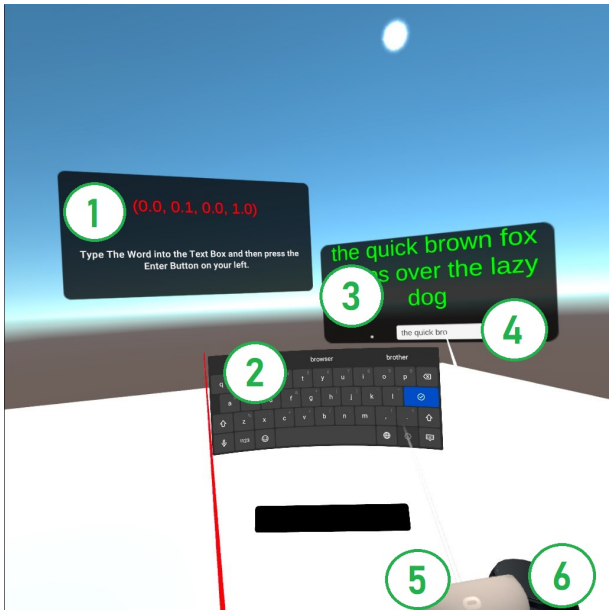


Figure 7: Data collection app. (1) The background app records the sensor readings even it is not in focus. The user types into the foreground keyboard (2), following the sentence prompt (3), in the text field (4). (5) The system hand controller is used to type, but the background app only has access to the (6) application hand controller, which is frozen while the keyboard is in focus.

in Fig. 7. The app prompts the user to type the specified words in the text field using the default Meta Quest 2 system keyboard. The app records the headset's accelerometer and gyroscope data (velocity and angular velocity) at 72 Hz [56]. If users glance up at the text prompt while typing, this adds noise to the collected data, although we did not notice such movements when visually inspecting the data. During the training phase, our data collection app has ground truth access to the typed characters for analysis and training the machine learning models [13]. During a real attack, the text input is to the foreground application, and the malicious background app only has access to the headset tracking readings and the pre-trained machine learning models.

User study recruitment and warm-up phase. With approval from our institution's IRB, we collected typing data from volunteers wearing an AR/VR headset. The user study was performed with 21 participants of varying age, height, weight, gender, and amount of experience with VR. This is in line with prior AR/VR human subjects research [29, 36, 52], which have 2-25 volunteers per experiment. Users were recruited through Slack or personal contacts, requesting volunteers for an AR/VR typing experiment, and participants were entered into a raffle for a $50 Amazon gift card. Before starting the study, users were informed that "the headset will record signals from your behavior while you are interacting with it" and that they could stop the study at any time. Users were initially not informed of the exact purpose of the study in order to avoid "participant bias" [8], an effect where participants who know the hypothesized outcome of a study may act to achieve (or confound) the outcome. Users were also instructed on how to operate the headset and controllers, how to adjust the headset to fit comfortably, and not to touch the headset with one's hands or anything else that would interfere with the headset's tracking. Each trial took up to 30 minutes, varying based on the individual participant's natural typing speed.

User study experiment phases. Volunteers participated in two phases of data collection: *word typing* and *sentence typing*. Each phase lasted approximately 15 minutes with a break in between. The former was used to train and test the word and character pair classifiers. The latter was used to train and test the sentence segmenter as well as provide additional words and character pairs for the classifiers.

- *Word typing phase:* A participant wore the VR headset and held the right hand controller while typing each prompted word and pressing the submit button between each word. The prompted words were from a list of 40 different words of 2 or 6 characters in length, selected at random from the top 5000 most frequent words in the Corpus of Contemporary American English [11]. For ease of ground truth labeling in this experiment, the submit button was placed far away to the side from the keyboard and text field, in order to require the user to move their head a large amount in between words and create a large signal change on the gyroscope readings. Each participant repeated this process for all 40 words, 3 times each (120 words per participant). In total, 21 users participated with 2520 total typed words.

- *Sentence typing phase:* Participants typed full sentences, with words separated using the space bar (rather than the submit button in the previous experiment, in order to be more realistic). The sentences were 5 words long and were randomly generated permutations from the 2-letter words from the word typing phase plus 20 new 6-letter words. The participant typed the full sentence, cleared the text field with the submit button, and each word was represented 3 times in the total sentences for each participant. 21 participants participated in this experiment for a total of 610 sentences typed.

We combined the unique words from the word and sentence typing phases when performing word classification, for a total of 60 unique words and 5040 samples. Over all experiments, about 6% of words contained typographical errors and were therefore not classifiable. These errors were not evenly distributed between participants and varied from 2 to

12% depending on the participant. Typos were included in the character pair classifier if they appeared at least once per participant.

User post-study disclosure. Several weeks after the study, users were sent a debrief email disclosing the purpose of the study, the specific nature of the data collected (*i.e.,* headset gyroscope and accelerometer data), and researcher contact information in case of questions, concerns, or complaints. We acknowledge that the debrief should have taken place in person immediately after after the study. Users did not express or appear to experience any discomfort during the study, and we did not receive any complaints or requests to exclude a participant's data after the debrief messages were sent.

4.2 Comparison Methods

We compared our Classifier against several other methods: a KNN-based approach [23] and a Random Convolution Kernels transform (ROCKET) with logistic regression [12].

- **KNN:** We experimented with multiple distance metrics for the KNN, including including Dynamic Time Warping [7] and first order statistics of the time series sample (*e.g.,* mean, variance, etc.); however, neither of these metrics gave prediction results significantly better than random. Instead, we element-wise multiply the head tracking vector (6 elements) by the inter-frame time and sum them. This essentially performs a coarse integration of the acceleration and angular velocity into a single velocity-pose vector. The number of clusters was set to the number of classes in each problem.
- **ROCKET:** The main idea of ROCKET is to convolve the time series with 10,000 randomly generated kernels to produce features, and train a logistic regression on these features. There has been some success using ROCKET in classifying 6 DoF data into activities [49]. We chose ROCKET as a baseline due to its past success and its ability to handle multivariate time series data.
- **Random:** We compute the probability of a random guess, *i.e.,* $1/N$, where N is the number of classes to predict (*e.g.,* the corpus size for word classification).

We use the ROCKET implementation in the `sktime` package [3] and KNN from `scikit-learn` [1]. The CNN model was implemented using Tensorflow 2.8.0 and the Adam optimizer [14]. All methods were coded in Python 3.7.0 [2] and run on a PC with a 2.8 GHz Intel i7 processor and 32 GB of RAM, taking up to 1 hour to train a CNN. Unless otherwise noted, 75% of the data was used for training, and 25% for test. Results conducted with our full dataset (5040 typed words, 60 unique words) are denoted by the method name appended with a "+" (*e.g.,* "CNN+"). Results obtained from an initial, smaller dataset (1200 typed words, 40 unique words) lack the suffix (*e.g.,* "CNN").

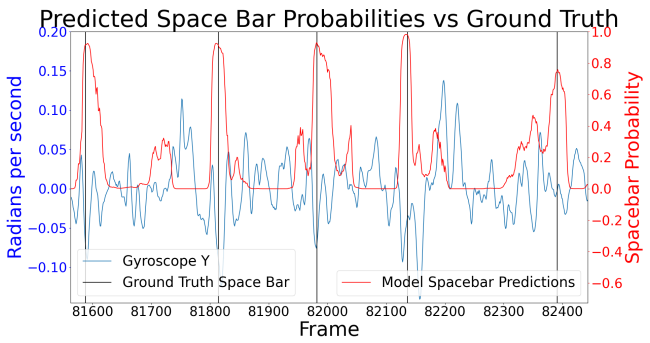


Figure 8: Example of TyPose's predicted word boundary probabilities (red) plotted against the ground truth (black). The raw gyroscope trace is shown in blue.

Data pre-processing. Before feeding in the gyroscope and accelerometer data into the Segmenter, we perform several pre-processing steps. Consumer-grade accelerometer and gyroscope data can be noisy [25]. An optional pre-processing smoothing pass may be warranted, which is performed by setting the accelerometer and gyroscope values at a specific frame to the average of the surrounding values (windowed averaging). TyPose performs this pre-processing step for the Segmenter, but not for the Classifier, as it was not found to have a significant impact there.

4.3 Evaluation of Segmenter

4.3.1 Sentence Segmentation into Words

Setup. We use the sentence typing dataset (Section 4.1) to train and evaluate our boundary classifier. For Scenario 1A, 487 sentences were used to train and 54 were used for evaluation. For Scenario 1B with a single victim, 23 sentences were used for training and 6 were used to evaluate. To handle the class imbalance, since there were few examples of space bar presses, we oversampled the minority class and added class weights.

To evaluate word classification, we walk a window of size $W = 200$ frames across an un-labeled time series trace, including all 6 DoF. Fig 8 shows an example trace with the probability of each window being a boundary marked in red, and the vertical black lines indicating ground truth boundaries. The windows are overlapping in the test set, so ideally only those windows with a boundary near the center will be classified as highly likely to contain a word boundary. If several adjacent windows have softmax probability $> T = 0.8$, the local maximum is predicted as the boundary. A predicted boundary is considered as a true positive if its center is within

		Boundary Predicted?	
		Yes	No
Actual	Yes	TP = 69	FP = 154
	No	FN = 135	TN = 83, 067

Table 2: Sentences segmented into words (by space bar presses) across multiple users in Scenario 1A.

		Boundary Predicted?	
		Yes	No
Actual	Yes	TP = 14	FP = 21
	No	FN = 10	TN = 9633

Table 3: Sentences segmented into words (by space bar presses) for 1 participant in Scenario 2A.

10 frames of the true boundary. This definition is used to compute the true positives, false positives, false negatives in the following results.

Results. For Scenario 1A, where multiple users' data is used to train a model, the CNN-based sentence segmenter results in 69 true positives, 154 false positives, and 135 false negatives out of 223 true space bar presses, as shown in Table 2. When trained on sentences from multiple participants, TyPose is able to find word boundaries for many words, albeit with a fairly high false positive rate. We account for this later in the end-to-end attack (§5) using data augmentation strategies and information about the average word length.)

For Scenario 1B, where only data from a specific victim is used for training, and later test, the CNN classifier gave 21 false positives and 10 false negatives, as shown in Table 3. The performance seems better when using data from just one participant, out-performing training and evaluation on data on many individuals, which makes sense intuitively, since the segmenter becomes personalized to an individual user.

4.3.2 Sentence Segmentation into Characters

We also utilize the sentence typing data to train and evaluate Scenarios 2A and 2B. Instead of training on windows centered on just space characters, we train on all character entries. The window size was shrunk to 32 frames due to the samples being closer together in time. When training with 19 users and predicting the remaining 2 users, the classification accuracy is 71%. When training and testing on a single user (with an 80/20 train/test split), the classification accuracy is 85%. Since the focus of this paper is on word classification, we describe those results next.

Method	Top-1 accuracy	Top-5 accuracy
Random	0.05	0.25
ROCKET	0.18	0.66
CNN	0.75	0.99
CNN+	0.65	0.92

Table 4: Word classification accuracy in Scenario 1B, when training and testing from 1 participant.

4.4 Evaluation of Classifier

4.4.1 Word Classification

Setup. For Scenario 1A, we used the *word typing* dataset described earlier (Section 4.1), which contains both two-letter and six-letter words. For ROCKET and CNN, 75% of the data were used for training and 25% for evaluation for both word lengths (corresponding to 408 two-letter words and 398 six-letter words for training, and 136 and 132 for test respectively). Additional data were gathered for CNN+ where 19 users were used for training and 2 users excluded for testing, then 5-fold cross validated (5 different train/test splits) for the final averaged accuracy results. For Scenario 1B, we form a third word classification data set from additional trials performed by a random participant to evaluate ROCKET and CNN. The participant repeated the experiment three times on three separate days for a total of 180 six-letter words. For CNN+, five participants typed a total of 120 six letter words each out of 40 possible instead of 20 words and were cross validated.

Results. For Scenario 1A where all users' data is used for training, Fig. 9a shows the top-1 and top-5 accuracy for the across all 2-letter and 6-letter words, for all the different methods (CNN+, CNN, ROCKET, and Random). Top-*k* accuracy is a common evaluation metric for speech and keyboard inference [29, 36, 52]. While both CNN and ROCKET drastically out-perform random guessing, the CNN proves the most accurate overall. Focusing in on the prediction accuracy of the 6-letter words along (Fig, 9b), we see the accuracy can be even better than the general case. This may be because the 6-letter words have longer duration, providing more information to the classifier. Adding additional users' data and additional words (CNN+) has a slightly increased accuracy. These results are also in line with other top-*k* accuracies reported by other cross-modality AR/VR inference attacks [52]. Note that we do not include KNN results, which were very poor. This is because the KNN input features only represented the aggregate head movement during the word, in order to have a low-dimensional (6×1) input size and a reasonable run time; however, higher-dimensional input features would be needed to accurately differentiate words from each other.

For Scenario 1B, the top-1 and top-5 accuracy for the different methods are shown in Table 4. The classification ac-

Method	Top-1 accuracy	Top-5 accuracy
Random	0.025	0.125
ROCKET	0.289	0.675
CNN	0.353	0.710
CNN+	0.400	0.820

(a) Classify from 2-letter or 6-letter words

Method	Top-1 accuracy	Top-5 accuracy
Random	0.05	0.25
ROCKET	0.390	0.796
CNN	0.654	0.932
CNN+	0.659	0.929

(b) Classify from 6-letter words only

Figure 9: Word classification accuracy in Scenario 1A, when training and testing on data from all participants.

Method	Top 1 accuracy	Top 5 accuracy
Random	0.022	0.111
KNN	0.18	0.48
ROCKET	0.20	0.54
CNN	0.23	0.58
CNN+	0.33	0.72

Table 5: Character pair classification accuracy in Scenario 2A, across all participants.

Method	Top-1 accuracy	Top-5 accuracy
Random	0.016	0.08
KNN	0.21	0.50
ROCKET	0.24	0.52
CNN	0.28	0.58
CNN+	0.31	0.77

Table 6: Character pair classification accuracy in Scenario 2B, out of 62 (121 for CNN+) possible character pairs, from 1 participant.

curacy is similar to that of Scenario 1A (Fig. 9a), again with the CNN-based methods performing the best. Adding 20 additional words and cross validating the single user results brings the accuracy in line with the multi user scenario. We did note, however, that some users have results far above (or below) the average, suggesting that some participants have more predictable head movements than others. Hence models trained on a single participant may be able to learn participant-specific behaviors and exploit that learning for improved classification accuracy. On the downside, Scenario 1B may be less practical because it requires ground truth training data from a target victim.

4.4.2 Character Pair Classification

Setup. For Scenario 2A, we further subdivide the words from the *word typing* dataset in their constituent character pairs. For example, the word "fox" contains two character pairs, "fo" and "ox". In total, 1852 character pairs were used for training and 618 were used for test when evaluating KNN, ROCKET, and CNN. This resulted in 45 possible character pairs. For CNN+, 121 unique character pairs were used for a total of 9448 training and 2362 test sample character pairs. For Scenario 2B, similar to word classification (Section 4.4.1), we formed a character pair dataset from a single user. The number of possible character pairs (62) is larger than in Scenario 2A dataset, due to a large number of frequently repeated typographical errors.

Results. For Scenario 2A, we show the top-1 and top-5 accuracy in Table 5. The CNN-based outperforms other methods and is far better than random, but further accuracy improvements are desirable. Adding additional users and samples to the training set improves the accuracy by 25-50% (CNN+ compared to CNN). We believe that even noisy predictions could still be useful in reducing the search space of possible passwords, for example by passing in these character pair probabilities to password cracking software such as "John the ripper" [41]. Reducing the search space would reduce the password cracking time and show headset accelerometer and gyroscope as a useful side-channel for password stealing.

For Scenario 2B that zooms on a single user, we show the top-1 and top-5 accuracy in Table 6. Compared to Scenario 2A that uses all users' data for training, the personalized attack has a better accuracy. This is in line with the results in previous subsections.

5 Demonstration of end-to-end attack

In the event that the attacker does not have access to word or character pair entry times, an end-to-end attack on the entire unmarked time series of the victim's head pose is needed. Towards this, we combine the models from Sections 3.2 and 3.3 to explore the feasibility of an end-to-end attack.

Setup. We utilize the sentence typing dataset from Section 4.1. Following the top branch in Fig. 5, the raw sensor streams are fed into the segmenter, which finds the word boundaries that are then fed into the Word Classifier. To measure the performance of the end-to-end attack, we compute several metrics:

- The **edit (Levenshtein) distance** $e(A,B)$ assigns a penalty of 1 to every add, delete, or swap made to trans-

form the ground truth sentence A into the predicted sentence B [18, 36]. The edit distance is the minimum number of such operations needed, and lower is better. The minimum edit distance is 0 and the maximum is unbounded.

- The **normalized edit distance**, $\frac{e(A,B)}{\max(|A|,|B|)}$, normalizes the edit distance by the maximum sentence length, since sentence lengths are variable.
- The **discounted edit distance** assigns a lesser penalty to swaps when the true word is within the top 5 predictions. The weight is [0.2, 0.4, 0.6, 0.8] if the true word is predicted as the [2nd, 3rd, 4th, 5th] most likely word, respectively.

To account for the noisy segmenter, we employed two strategies. First, we performed data augmentation when training the Word Classifier; specifically, we add a random amount (up to 1 second) of extra data to the beginning and end of the true word segment. Second, we provided the average word and sentence duration as side information; with an average word length L and sentence typing duration D, the segmenter picks the $\lfloor \frac{D}{L} \rfloor - 1$ most likely word boundaries in a sentence. The results reported below are the average of 5-fold cross-validation. Each test has about 50-60 sentences from never before seen users, so the scenario is quite challenging.

Results. The edit distance is 4.6, the discounted edit distance is 4.4, and the normalized edit distance is 0.93. A naive attacker could guess $\lfloor \frac{D}{L} \rfloor \approx 5$ words in a sentence with a 1/40 chance of getting each word correct, giving a naive edit distance at 4.875. Thus compared to a random guess, our approach has better performance. We observe that the end-to-end attack is sensitive to the segmenter's performance, since it is the first step of the end-to-end pipeline. Upon decomposing the edit distance, we find that 16% are from insertions, 7% are from deletions, and 77% are from swaps. The insertions and deletions can be attributed to segmenter errors, while the swaps could be attributed to the segmenter or classifier.

The end-to-end attack results could likely be improved by adding priors on English grammar semantics. However, our scenario is a particularly challenging one with users typing random sequences of words to form sentences, so we could not experiment with such priors. Our approach could also be combined with other sensor modalities to improve segmentation, such as performance counters [64] or WiFi signals [4]. More sophisticated post-processing, such as a feedback loop between the word classifier and sentence segmenter, could also help. Overall, we believe that this end-to-end attack is a good starting point to demonstrate the feasibility of head pose as a source of information leakage for hand-typed sentences. Moreover, each of the individual components of the end-to-end attack can be used independently; for example, if the attacker desires only a single word to be detected (*e.g.*, the answer to a secret question password challenge), then the word classifier alone can be used with good accuracy. Further discussion is provided in the Limitations section.

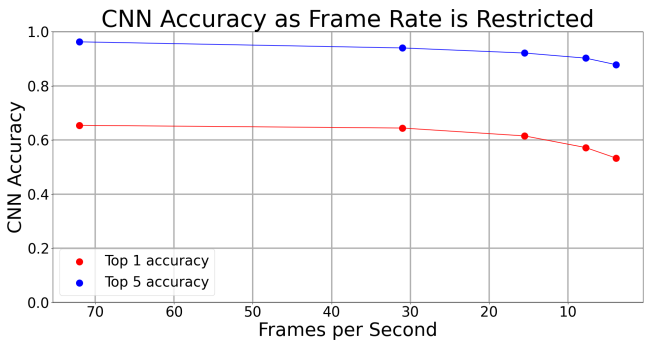


Figure 10: Reducing the rate that head tracking information is given to the background application does not have significant effects on Classifier accuracy until < 10 Hz.

6 Attack Mitigation

The simplest way to prevent the system's attack is to disallow background apps' access to the VR headset accelerometer and gyroscope readings, when the background app is out of focus. However, this may not be desirable as it also prevents the background app from updating the rendered image in response to the user's movements (in other words, causing a "freeze"), leading to poor user experience [24, 42]. Therefore, we first investigate mitigation strategies that try to avoid harming the user experience by still allowing background app rendering. We experiment with two methods: (a) reducing the frequency that the sensor streams are given to the background app, and (b) reducing the precision of the floating point values provided to the background app.

Reduced IMU sampling rate. We re-train and evaluate the CNN from Section 3.2 on word classification using the same dataset as Fig. 9b, but down-sample the IMU reading provided to the background app. We plot the classification accuracy in Fig. 10 for various sampling frequencies, ranging from 72 Hz (the default) to 5 Hz. The top-1 and top-5 accuracies do not drop much as the sampling frequency decreases to 5 Hz. This suggests that 5 Hz still supplies sufficient information for classification. Alternatively, the robustness to sampling rate may suggest that an important predictor of what a user is typing is the length in time of the sample (although we experimented with length alone as a classifier, and found it insufficient). In any case, we conclude that the frequency reduction needed (to less than 5 Hz) would also inhibit the background app's ability to update the display at an acceptable frame-rate [69].

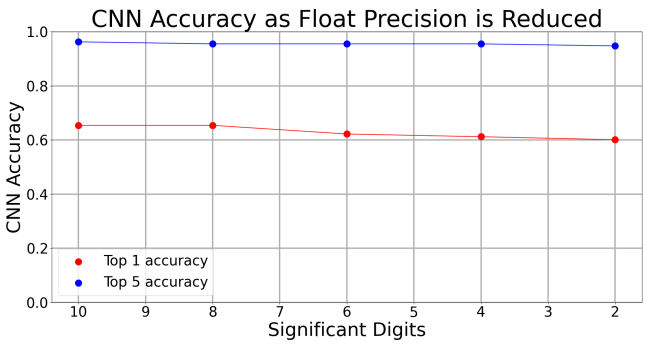


Figure 11: Rounding off the floating point values of the head tracking information given to the background application does not have significant effects on CNN accuracy.

Reduced precision. We also re-train and evaluate the CNN with the same data as above, but during pre-processing, we round off the floating point IMU values to different decimal precision. Plotting the results in Fig. 11, we see minimal accuracy reduction even at as low as two significant figures. This is likely because even though information from the significand of the floating point number is reduced, the remaining information along with the exponent, sign, and in the length of the samples provides enough for classification. Since reduction of significant digits in head tracking information can lead to "judder" or drift of the rendered image, without significantly reducing the classification accuracy, we do not recommend this as a viable mitigation.

Overall, given that the classification can still be performed at better than random accuracy with both of these mitigation techniques, more elaborate defense mechanisms are needed. One possibility is for the AR platform to move the user to a default "system room" in the background whenever a keyboard is opened in the foreground, instead of keeping the malicious app in the background. This comes at the expense of reducing user immersion and possibly losing state in the background app. Another possibility is to randomize the keyboard location or size each time it is opened, or even randomize digit key locations in the case of PIN code entering. Alternatively, the AR platform may employ a modified version of "time warping" [24] where the background app is rendered with a wider field-of-view, and the AR platform later crops the final rendered image using real-time head tracking data not provided to the background app. The above mitigation strategies may degrade user experience, and further human subjects research may be needed to understand their perceptual impacts.

7 Limitations

While participating in the user study, some volunteers who were not able to find a comfortable way to wear the headset did not visibly move their heads at all while typing, possibly due to discomfort. The data from these users were not discarded but did show less head motions than others. This suggests that if a user were aware of the possibility of malicious head tracking, it is possible to hold one's head nearly still while typing in order to confuse the classification model and limit its accuracy. However, this would require conscious behavioral change on the part of the users.

A second limitation is that any change to the text entry method may require the collection of a new training dataset and model in order to maintain TyPose's accuracy. These changes could include operating system updates that replace the system keyboard, the addition of a numberpad, or future novel AR/VR text entry mechanisms. Related to this, the machine learning models we experimented with may be suboptimal, as the primary goal of this project was to demonstrate the attack's feasibility. We would be interested to see other learning-based approaches towards these types of classification problems. Further, the models are trained to detect only 60 unique words, and it could fail if users type a more diverse set of words. The attack could be generalized by expanding this dictionary through additional user data collection and model training. The set of words could also be carefully selected so that the attack could still recognize key words in the sentence to extract the main meaning, even if every word is not perfectly recognized.

Finally, a key lesson learned is the cascading effects of errors and thus the importance of the first segmentation step on end-to-end performance. The classifier accuracy dropped when noisy segmentation boundaries were provided by the segmenter, despite our best data augmentation techniques to mitigate this. Since segmentation has shown success in single-modality attacks [29, 36], we are optimistic that improved segmentation techniques or additional side channels [4, 64], combined with the already high-performing classifier, can help the end-to-end attack.

8 Related Work

AR/VR key-logging attacks. AR and VR devices open the door to new application spaces with unique threat models. As a result, a number of new attacks that target these devices have emerged. Arafat et al. [4] developed a key logging side channel attack leveraging fluctuations in wireless signal around a VR user. Different head and hand motions cause different fluctuations in the Wi-Fi signal which can be correlated to key inputs. Similarly, TyPose uses changes in head movements and timings to infer user text input, but does not require external wireless sensors. Face-Mic [52] is an example of another type of a cross modality attack (head tracking to infer spoken words), but focuses on audio input whereas text-based input is currently more common.

Meteriz-Yildiran et al. [36] and Hologger [29] demonstrate single modality attacks capable of stealing sensitive information on AR devices (*i.e.,* hand tracking to infer hand-

based typing, head tracking to infer head-based typing), rather than cross modality (head tracking to infer hand-based typing) as we do. Our problem has different threat models: We did not use hand tracking data as in [36] because it is blocked from background apps by popular AR/VR development engines like Unity, in our experience; and head-based typing as in [29] is less common and results in higher typing error rates for users compared to hand-based typing [17]. Our problem is significantly more challenging than these single-modality attacks because we have to infer both when and what words are typed, causing errors to accumulate in the end-to-end attack. Such single-modality attacks get the latter for free (*e.g.,* once you can correctly estimate when a hand presses a key, identifying which key is relatively straightforward from the hand pose).

AR/VR authentication mechanisms. AR/VR introduces unique password entry methods [22, 31, 53, 61]. These works use unique 3D hand motion paths, head gaze, or eye tracking to enter passwords, suggesting that if the sensor data used for these input methods were logged, the sensitive information could be inferred. These are outside the scope of this work, as TyPose focuses on text entry using virtual keyboards. There have also been efforts to design shoulder surfing resistant authentication methods [40, 63]. Since TyPose uses head tracking information freely given to AR/VR applications, shoulder surfing is not necessary and mitigating it would not reduce the efficacy of our attack.

Other AR/VR security issues. Other security and privacy threats on AR/VR devices and applications are covered by several excellent surveys [39, 47, 48]. Our work falls at the intersection of data access, input security, and multiple applications mentioned by these surveys. Cheng et al. [9] investigate the human impacts of AR/VR perceptual manipulation attacks. Shang et al. [51] use network traffic analysis in a multi-user app to infer user location through custom built malicious applications. Our attack focuses on inferring sensitive text rather than location. Designing sharing techniques to enforce permissions or prevent 3rd parties from accessing private virtual content is another problem [44, 50], as is malicious output of the AR display [26]. TyPose is orthogonal to such works in that it focuses on AR/VR input modalities, rather than rendering and sharing of virtual content.

Conventional key-logging side-channel attacks. Our work investigates head movement information to build up side-channel for information leakage attacks. Prior works have explored using motion sensor to steal keylogging secrets on mobile phones [62] and smart watches [28, 30, 59, 60, 62]. Similar to the insights in our attack, the user motion as they type correlates with the location of the keys on a soft keyboard, enabling these attacks. Other works exploited acoustically [5, 16] or EM [6, 58] side-channels to extract keylogging input. However, these attacks require physically connecting a probe or microphone close to the keyboard. In more conventional computing settings, several works have deployed keystroke inference attacks based on CPU-based [46], cache-based [15] or GPU-based [37] side-channel. More generally, the use of machine learning in keylogging attacks using data from seemingly-unrelated sensor inputs has some precedent [38, 68]. In particular, prior knowledge of the English language and grammar [68] could be used to improve on our results.

9 Conclusions

As AR/VR devices become prevalent, there is a pressing need for research into their security and privacy risks. In this work, we show an attacker may freely obtain a stream head tracking data from an AR/VR device, segment it, and classify it in order to obtain sensitive text information. The attack is shown to be especially accurate when trained on specific targeted victims. While a simple mitigation tactic – blocking access to the head tracking data – exists, it breaks desired functionality in a background application. The attack is also resilient to less extreme mitigation strategies, such as reducing the frequency and precision of the sensor readings, to the point that the background app would have to be visually compromised before successful attack mitigation. In future work, we plan to characterize user sensitivity to the background app's visual display, in order to develop new mitigation strategies, as well as other text entry methods.

Acknowledgements

We thank the anonymous reviewers and shepherd for their valuable comments, from which this paper greatly benefited. We are also grateful to the user study participants who generously volunteered their time. This work was partially supported by the NSF grants CNS-1942700, CNS-2053383, CCF-2212426, and a Meta faculty research award.

References

[1] scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/, 2007.

[2] Python release 3.7.0. https://www.python.org/downloads/release/python-370/, 2018.

[3] sktime. https://www.sktime.org/en/stable/, 2019.

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

[5] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy*, 2004.

[6] Andrea Barisani and Daniele Bianco. Sniffing keystrokes with lasers/voltmeters. *Proceedings of Black Hat USA*, 2009.

[7] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 1994.

[8] Caridad F Brito. Demonstrating experimenter and participant bias. *Activities for teaching statistics and research methods: A guide for psychology instructors*, 2017.

[9] Kaiming Cheng, Jeffrey Tian, Tadayoshi Kohno, and Franziska Roesner. Exploring user reactions and mental models towards perceptual manipulation attacks in mixed reality. In *USENIX Security*, 2023.

[10] CNBC. Coronavirus could be catalyst to reinvigorate virtual reality headsets. https://www.cnbc.com/2020/05/02/coronavirus-could-be-catalyst-to-reinvigorate-virtual-reality-headsets.html, 2020.

[11] Mark Davies. The corpus of contemporary american english as the first reliable monitor corpus of english. *Literary and linguistic computing*, 25(4):447–464, 2010.

[12] Angus Dempster, François Petitjean, and Geoffrey Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34, 09 2020.

[13] Google. Logcat, Android Developers. http://android-doc.github.io/tools/help/logcat.html, 2022.

[14] Google. Tensorflow releases. https://github.com/tensorflow/tensorflow/releases, 2022.

[15] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+ flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016.

[16] Tzipora Halevi and Nitesh Saxena. Keyboard acoustic side channel attacks: exploring realistic and security-sensitive scenarios. *International Journal of Information Security*, 14(5):443–456, 2015.

[17] John Paulin Hansen, Kristian Tørning, Anders Sewerin Johansen, Kenji Itoh, and Hirotaka Aoki. Gaze typing compared with input by head and hand. In *ACM Symposium on Eye tracking research & applications*, 2004.

[18] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[19] Kasirat Turfi Kasfi, Andrew Hellicar, and Ashfaqur Rahman. Convolutional neural network for time series cattle behaviour classification. In *ACM Workshop on Time Series Analytics and Applications*, 2016.

[20] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. Segmenting time series: A survey and novel approach. In *Data mining in time series databases*, pages 1–21. World Scientific, 2004.

[21] Kickstarter. Oculus Kickstarter. https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game, 2015.

[22] Manu Kumar, Tal Garfinkel, Dan Boneh, and Terry Winograd. Reducing shoulder-surfing by using gaze-based password entry. In *USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2007.

[23] J. Laaksonen and E. Oja. Classification with learning k-nearest neighbors. In *International Conference on Neural Networks (ICNN)*, 1996.

[24] Steven LaValle. Virtual reality. *Cambridge University Press*, 2016.

[25] Steven M. LaValle, Anna Yershova, Max Katsev, and Michael Antonov. Head tracking for the oculus rift. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[26] Kiron Lebeck, Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner. Securing augmented reality output. In *IEEE Symposium on Security and Privacy (SP)*, 2017.

[27] Chien-Liang Liu, Wen-Hoar Hsaio, and Yao-Chung Tu. Time series classification with multivariate convolutional neural network. *IEEE Transactions on Industrial Electronics*, 66(6):4788–4797, 2019.

[28] Xiangyu Liu, Zhe Zhou, Wenrui Diao, Zhou Li, and Kehuan Zhang. When good becomes evil: Keystroke inference with smartwatch. In *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[29] Shiqing Luo, Xinyu Hu, and Zhisheng Yan. Holologger: Keystroke inference on mixed reality head mounted displays. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2022.

[30] Anindya Maiti, Oscar Armbruster, Murtuza Jadliwala, and Jibo He. Smartwatch-based keystroke inference attacks and context-aware protection mechanisms. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2016.

[31] Mehrube Mehrubeoglu and Vuong Nguyen. Real-time eye tracking for password authentication. In *IEEE International Conference on Consumer Electronics (ICCE)*, 2018.

[32] Meta. Education | Oculus. https://www.oculus.com/experiences/go/section/161391067688077/.

[33] Meta. Workrooms | VR for Business Meetings. https://www.oculus.com/workrooms/.

[34] Meta. Accessibility improvements, and air link for quest 1 in the latest oculus software update. https://www.oculus.com/blog/multitasking-accessibility-improvements-and-air-link-for-quest-1-in-the-latest-oculus-software-update/, 2021.

[35] Meta. Latest Oculus Quest Update Fosters Developer Creativity With App Lab and Connects People With Messenger . https://www.oculus.com/blog/latest-oculus-quest-update-fosters-developer-creativity-with-app-lab-and-connects-people-with-messenger/, 2021.

[36] Ülkü Meteriz-Yıldıran, Necip Fazıl Yıldıran, Amro Awad, and David Mohaisen. A keylogging inference attack on air-tapping keyboards in virtual environments. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2022.

[37] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. Rendered insecure: Gpu side channel attacks are practical. In *ACM conference on computer and communications security (CCS)*, 2018.

[38] Sashank Narain, Amirali Sanatinia, and Guevara Noubir. Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning. In *ACM conference on Security and privacy in wireless & mobile networks (WiSec)*, 2014.

[39] Naheem Noah, Sommer Shearer, and Sanchari Das. Security and privacy evaluation of popular augmented and virtual reality technologies. In *IEEE International Conference on Metrology for eXtended Reality, Artificial Intelligence, and Neural Engineering*, 2022.

[40] Ilesanmi Olade, Hai-Ning Liang, Charles Fleming, and Christopher Champion. Exploring the vulnerabilities and advantages of swipe or pattern authentication in virtual reality (vr). In *ACM International Conference on Virtual and Augmented Reality Simulations*, 2020.

[41] Openwall. John the Ripper. https://github.com/openwall/john, 2013.

[42] Bruno Patrão, Samuel Pedro, and Paulo Menezes. How to Deal with Motion Sickness in Virtual Reality. In Paulo Dias and Paulo Menezes, editors, *22o Encontro Português de Computação Gráfica e Interação 2015*, 2020.

[43] Bin Qian, Yong Xiao, Zhenjing Zheng, Mi Zhou, Wanqing Zhuang, Sen Li, and Qianli Ma. Dynamic multi-scale convolutional neural network for time series classification. *IEEE Access*, 8:109732–109746, 2020.

[44] Shwetha Rajaram, Franziska Roesner, and Michael Nebeling. Designing privacy-informed sharing techniques for multi-user ar experiences. *International Workshop on Security for XR and XR for Security (Vr4Sec)*, 2021.

[45] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[46] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *ACM conference on Computer and communications security (CCS)*, 2009.

[47] Franziska Roesner and Tadayoshi Kohno. Security and privacy for augmented reality: Our 10-year retrospective. In *International Workshop on Security for XR and XR for Security (VR4Sec)*, 2021.

[48] Franziska Roesner, Tadayoshi Kohno, and David Molnar. Security and privacy for augmented reality systems. *Communications of the ACM*, 57(4):88–96, 2014.

[49] Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and Anthony Bagnall. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery (*, 35(2):401–449, 2021.

[50] Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner. Secure multi-user content sharing for augmented reality applications. In *USENIX Security Symposium*, 2019.

[51] Jiacheng Shang, Si Chen, Jie Wu, and Shu Yin. ARSpy: Breaking Location-Based Multi-Player Augmented Reality Application for User Location Tracking. *IEEE*

Transactions on Mobile Computing, 21(2):433–447, 2022.

[52] 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 *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2021.

[53] S. Stephenson, B. Pal, S. Fan, E. Fernandes, Y. Zhao, and R. Chatterjee. Sok: Authentication in augmented and virtual reality. In *IEEE Symposium on Security and Privacy (SP)*, 2022.

[54] Ivan E Sutherland. A head-mounted three dimensional display. In *Proceedings of the fall joint computer conference*, pages 757–764, December 1968.

[55] Rahmadi Trimananda, Hieu Le, Hao Cui, Janice Tran Ho, Anastasia Shuba, and Athina Markopoulou. OVRseen: Auditing Network Traffic and Privacy Policies in Oculus VR. In *USENIX Security Symposium*, 2022.

[56] Unity. Scripting API: CommonUsages - Unity - Manual. https://docs.unity3d.com/ScriptReference/XR.CommonUsages.html.

[57] Unity. Unity version 2020.3.26f1 download archive. https://unity3d.com/get-unity/download/archive, 2020.

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

[59] Chen Wang, Xiaonan Guo, Yan Wang, Yingying Chen, and Bo Liu. Friend or foe? your wearable devices reveal your personal pin. In *ACM on Asia conference on computer and communications security (ASIACCS)*, 2016.

[60] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. Mole: Motion leaks through smartwatch sensors. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2015.

[61] Waqas Wazir, Hasan Ali Khattak, Ahmad Almogren, Mudassar Ali Khan, and Ikram Ud Din. Doodle-based authentication technique using augmented reality. *IEEE Access*, 8:4022–4034, 2020.

[62] Zhi Xu, Kun Bai, and Sencun Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2012.

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

[64] Yicheng Zhang, Carter Slocum, Jiasi Chen, and Nael Abu-Ghazaleh. It's all in your head(set): Side-channel attacks on ar/vr systems. In *USENIX Security*, 2023.

[65] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, 2017.

[66] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2012.

[67] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. *Network and Distributed System Security Symposium (NDSS)*, 2012.

[68] Li Zhuang, Feng Zhou, and J Doug Tygar. Keyboard acoustic emanations revisited. *ACM Transactions on Information and System Security*, 13(1):1–26, 2009.

[69] David J Zielinski, Hrishikesh M Rao, Mark A Sommer, and Regis Kopper. Exploring the effects of image persistence in low frame rate virtual environments. In *IEEE Virtual Reality (VR)*, 2015.