Computers & Security

VRKeyLogger: Virtual Keystroke Inference Attack via Eavesdropping Controller Usage Pattern in WebVR

--Manuscript Draft--

Manuscript Number:	COSE-D-22-00475
Article Type:	Full Length Article
Keywords:	Virtual Reality; WebVR; Web Security; Keystroke Inference; Virtual Keyboard; VR Controller Sensors; VR Side-Channel Attack
Corresponding Author:	Kilho Lee Soongsil University KOREA, REPUBLIC OF
First Author:	Jiyeon Lee
Order of Authors:	Jiyeon Lee
	Hyosu Kim
	Kilho Lee
Abstract:	WebVR is an emerging technology that allows users to experience VR (Virtual Reality) through typical web browsers, providing an integrated environment for various VR applications. One important problem of the VR technology is how to securely interact with users, in particular, implementing secure text input. A promising approach is to use a virtual keyboard rendered as a VR object. The VR user can enter certain text by clicking a sequence of virtual keys through the VR controllers, and the input text is handled in a secure way. However, despite the sensitivity of the input text, we found that there is a critical vulnerability that the VR controllers are not properly protected. The VR controller status can be disclosed to malicious entities, imposing a severe threat that an attacker's website can infer the input text by eavesdropping and analyzing the VR controller's movements. To accurately infer the input, the attacker should address two challenges: 1) determining which clicks correspond to the virtual keyboard and 2) identifying which key is pressed. In this paper, we propose a new keystroke inference attack framework, VRKeyLogger, that addresses such challenges with two key components: key-click classifier and key-click identifier. The key-click classifier effectively distinguishes clicks on the virtual keyboard based on the SVM classifier trained by the major features of the VR controller uses. The keyclick identifier then accurately identifies which key is pressed by transforming the clicked position into the local coordinate system of the virtual keyboard. We implemented a proof-of-concept prototype and conducted a user study with nine participants. In the extensive user study with three real-world WebVR applications, our VRKeyLogger results in classification and identification accuracy of 93.98 and 96.8% on average, respectively. This implies that the proposed attack poses a serious threat to WebVR security.
Suggested Reviewers:	

Cover letter

VRKeyLogger: Virtual Keystroke Inference Attack via Eavesdropping Controller Usage Pattern in WebVR

Jiyeon Lee, Hyosu Kim, and Kilho Lee

Dear Editors

We would like to submit the enclosed manuscript entitled "VRKeyLogger: Virtual Keystroke Inference Attack via Eavesdropping Controller Usage Pattern in WebVR" for possible publication in Journal of Computers & Security.

This article introduces a security flaw in WebVR where VR controllers are not properly protected from a website that has a different origin. It also introduces a novel attack technique for identifying users' key inputs by collecting the real-time status of VR controllers being rendered in the target VR scene. To infer the timing and location of key inputs, we designed two core components: a key-click classifier and a key-click identifier. The key-click classifier effectively distinguishes clicks on the virtual keyboard based on the SVM classifier trained by the major features of the VR controller uses. The keyclick identifier accurately identifies which key is pressed by transforming the clicked position into the local coordinate system of the virtual keyboard. With extensive evaluations, our attack demonstrated a high classification and identification accuracy of 93.98 and 96.8% on average, respectively, which outperforms other side-channel attacks with negligible performance overhead. To the best of our knowledge, this is the first study on the keystroke inference of virtual keyboards in the WebVR environment.

Would you please consider this manuscript for publication in your journal? If you have any questions or need any further information, please contact us via khlee.cs@ssu.ac.kr.

Thank you very much for your attention and consideration.

Best Regards

Kilho Lee

Jiyeon Lee.
Jiyeon Lee is a post-doc in the School of AI Convergence, Soongsil University, South Korea.
She received her BSc degree in Computer Science from Dankook University, and her MSc and PhD degrees in School of Computing from KAIST, South Korea.
Her research interests include general topics in security and privacy, web vulnerability research and cyber-physical systems.

Hyosu Kim.
Hyosu kim is an Assistant Professor in the School of Computer Science and Engineering, Chung-Ang University, Seoul, South Korea.
He received the B.S. degree from the Department of Computer Engineering, Sungkyunkwan University, and the M.S. and Ph.D. degrees from the School of Computing, KAIST, South Korea.
His research interests include cyber-physical systems, mobile systems, security, and privacy.

Kilho Lee.
Kilho Lee is an assistant professor in the School of AI Convergence, Soongsil University, South Korea.
He received his BSc degree in Information and Computer Engineering from Ajou University, and his MSc and PhD degrees in Computer Science from KAIST, South Korea.
His interests include system design for real-time embedded systems and cyber-physical systems. He won the best paper award of ACM MobiCom 2019 and IEEE CPSNA in 2014.

VRKeyLogger: Virtual Keystroke Inference Attack via Eavesdropping Controller Usage Pattern in WebVR

Jiyeon Lee[a], Hyosu Kim[b], Kilho Lee[a,*]

[a]*School of AI Convergence, Soongsil University, Seoul, 06978, Republic of Korea*
[b]*School of Computer Science and Engineering, Chung-Ang University, Seoul, 06974, Republic of Korea*

Abstract

WebVR is an emerging technology that allows users to experience VR (Virtual Reality) through typical web browsers, providing an integrated environment for various VR applications. One important problem of the VR technology is how to securely interact with users, in particular, implementing secure text input. A promising approach is to use a virtual keyboard rendered as a VR object. The VR user can enter certain text by clicking a sequence of virtual keys through the VR controllers, and the input text is handled in a secure way. However, despite the sensitivity of the input text, we found that there is a critical vulnerability that the VR controllers are not properly protected. The VR controller status can be disclosed to malicious entities, imposing a severe threat that an attacker's website can infer the input text by eavesdropping and analyzing the VR controller's movements. To accurately infer the input, the attacker should address two challenges: 1) determining which clicks correspond to the virtual keyboard and 2) identifying which key is pressed. In this paper, we propose a new keystroke inference attack framework, VRKeyLogger, that addresses such challenges with two key components: key-click classifier and key-click identifier. The key-click classifier effectively distinguishes clicks on the virtual keyboard based on the SVM classifier trained by the major features of the VR controller uses. The key-click identifier then accurately identifies which key is pressed by transforming the clicked position into the local coordinate system of the virtual keyboard. We implemented a proof-of-concept prototype and conducted a user study with nine participants. In the extensive user study with three real-world

*Corresponding author

WebVR applications, our VRKeyLogger results in classification and identification accuracy of 93.98 and 96.8 % on average, respectively. This implies that the proposed attack poses a serious threat to WebVR security.

Keywords: Virtual Reality, WebVR, Web Security, Keystroke Inference, Virtual Keyboard, VR Controller Sensors, VR Side-Channel Attack
PACS: 0000, 1111
2000 MSC: 0000, 1111

1. Introduction

With the rapid rise of the Metaverse, Virtual Reality (VR) has received a lot of attention as a key technology that will change our future lives. VR provides immersive user experiences, including enriched entertainment, education, healthcare, retail, and so on. As VR technology is applied to various domains, VR content has been developed on distinct VR platforms. Such fragmented VR platforms have hindered the wide dissemination of VR. To address this, VR comes to the Web, called WebVR. WebVR [1] is a new HTML5 standard that enables to run VR on top of typical web browsers. It offers distinct advantages, which are ease of development, deployment, and sharing of VR experiences with standardized interfaces. Due to these advantages, WebVR is supported by major commercial web browsers such as Firefox and Chrome.

One important issue in the development of VR technology is how to securely interact with users, in particular, implementing secure text input, as the text input may contain a vast amount of sensitive information (e.g., passwords or credit card numbers). In the VR environment, a user typically puts the text input through the virtual keyboard rendered in the VR scene, which may impose a risk of easily leaking the sensitive input to bystanders. A few studies have proposed such attacks. For instance, Ling et al. [2] introduced a vision-based attack that keeps track of the VR headset movements with a stereo camera and exploits them to infer the keystrokes. Another example is VR-Spy [3], which exploits the Channel State Information (CSI) of WiFi signals, which varies according to the user's virtual key input. Although such attacks demonstrate the potential of the keystroke inference threat, they focus on the native VR platforms.

Beyond the state-of-the-art, this paper focuses on a novel security threat caused by the nature of WebVR. We found that the WebVR virtual keyboard

Figure 1: Two browser windows are open on the user's computer. The VR controller information used in the left window is accessible in the right window.

imposes a critical vulnerability since the VR controllers are not properly protected in the WebVR environment when multiple websites try to access the VR devices. More specifically, when WebVR and attacker sites are opened on the same browser, the attacker site can eavesdrop on all the information of the VR controller used on the WebVR site without requiring any permission (see Figure 1). This is due to the absence of the proper data protection mechanism on the WebVR API supported by web browsers. Based on this vulnerability, we define the threat model as follows. We first introduce a *WebVR attacker* who infers the user's keystrokes made on the target WebVR site. The attacker entices users to visit the malicious website, which is controlled by the attacker. We assume that, while the user explores the target WebVR site, the attacker site is also opened as another browser window or an iframe element. In this situation, the attacker can obtain full access to the *Gamepad* object, which contains the pose and orientation information of the VR controller used on the target WebVR site. Note that obtaining the Gamepad object does not require any modification to the target site or additional app installation, as well as the attacker does not have to be a WebVR site. Consequently, such access to the Gamepad object allows the attacker to infer the user's keystrokes.

However, it is difficult to infer the exact text input because the Gamepad object only provides the raw data (i.e., position and orientation) of the VR controller. For effective key inference, the challenging issue is that the attacker should be aware of two important pieces of information: 1) when the user utilizes the virtual keyboard and 2) which virtual key is pressed by the user. We introduce `VRKeyLogger`, a novel keystroke inference attack framework that addresses such a challenge with two key components: *key-click*

classifier and *key-click identifier*. The key-click classifier builds an SVM-based classifier that reflects the differences in the VR controller status when a user makes *key-clicks* (i.e., clicks on the virtual keyboard) and *app-clicks* (i.e., other clicks); it then extracts key-clicks only. The key-click identifier calculates *hit-points* (i.e., the clicked position projected into the attacker's 2D space) for each key-click and transforms the hit-points into the local coordinate system of the target virtual keyboard. As the layout of the target virtual keyboard is publicly known, the attacker can effectively infer the exact keystroke made within the target WebVR site.

We implemented a proof-of-concept prototype of VRKeyLogger on top of the A-Frame WebVR framework [4], which is widely used and supported by commercial web browsers. With the prototype, we evaluated the performance of VRKeyLogger in terms of attack efficacy and overhead. We conducted user studies with nine participants to measure the success rates of our keystroke inference attack. Experimental results show that the proposed attack can successfully determine the key inputs with an average accuracy of 96.8% for three real-world VR apps, which outperforms the existing methods. In addition, VRKeyLogger shows a negligible FPS drop even when rendering a complicated target VR site, implying that VRKeyLogger can be effective in the wild.

To the best of our knowledge, this is the first study on the keystroke inference of virtual keyboards in the WebVR environment. In summary, our contribution is as follows:

- We first found a security flaw in WebVR where VR controllers are not properly protected from a website that has a different origin.
- We introduced a novel attack technique for identifying users' key inputs by collecting the real-time status of VR controllers being rendered in the target VR scene. To the best of our knowledge, this is the first study on the keystroke inference of virtual keyboards in the WebVR environment.
- To infer the timing and location of key inputs, we designed two core components: a key-click classifier and a key-click identifier.
- With extensive evaluations, our attack demonstrated a high classification and identification accuracy of 93.98 and 96.8% on average, respectively, which outperforms other side-channel attacks with negligible performance overhead.

This paper is organized as follows. Section 2 presents the WebVR background. Section 3 proposes the vulnerability, the threat model, and attack challenges. In Section 4, we introduce VRKeyLogger with two key components. Section 5 explains the proof-of-concept prototype implementation. Section 6 presents the evaluation design and results. Section 7 discusses the remaining issues, and Section 8 compares VRKeyLogger with related works. Section 9 concludes the paper.

2. Background

This section presents the background information, including terminology, the WebVR library, input methods, and virtual keyboards.

2.1. Terminologies

We first define the WebVR terminologies that we use throughout the paper. When a WebVR site is executed, it shows a *VR scene* that reflects the VR world. A user may change the execution mode of the WebVR site into the *VR mode*, which enables the user to explore the VR scene through HDM and VR controllers. Note that the WebVR site creates the *controller object* as an object in the VR world that corresponds to the VR controller. An *entity* refers to an object in a VR scene. In the VR world, every entity has its own position and orientation in coordinate systems. There are two types: the *global* and *local* coordinate systems [5] that represent the global space of the VR world and the local space of each object, respectively.

2.2. WebVR library

To implement a WebVR site, it is required for developers to utilize the complicated low-level WebGL APIs. To alleviate the complexity and ease of development, a number of JS (JavaScript) 3D libraries, such as *Three.js* [6] and *A-Frame* [4], provide a well-defined abstraction to implement a high-quality WebVR application. A-Frame is one of the representative WebVR frameworks introduced by Mozilla. Its compelling feature is that all entities to be rendered within a scene are defined by a markup language, which can be easily accessible via a Document Object Model (DOM) [7]. For instance, a developer can create a 3D box by defining a <a-box> tag in HTML and querying this entity via a JS method, such as `document.querySelector('a-box')`.

2.3. Input method

In the VR mode, users can interact with the WebVR using the *VR controller* [8]. A VR controller enables users to trigger various events on entities, such as clicks. Most WebVR frameworks provide a ray-casting system that renders laser rays with controller objects and makes it easier for users to click on some position of a target entity [9, 10]. To make a click, a user points the lay at a specific position and pulls the trigger of the controller; then, the lay may collide with some entity at a certain point (i.e., *hit-point*). We distinguish clicks into two categories: *key-click* and *app-click*. If a user makes a hit-point at a key on a virtual keyboard entity, such a click is classified as a key-click; otherwise, it is classified as an app-click. It is worth noting that there is another input method called *gaze cursors*. It generates a cursor indicating the focal point at which a user looks in the VR scene and makes a click event when the user gazes at the target entity for a while. This allows users to click something even without the VR controllers. However, it is very uncommon to use the virtual keyboard with the gaze cursor because it is very difficult to precisely point at a specific key. Therefore, this paper mainly focuses on the VR controller input.

2.4. Virtual keyboards

Most WebVR sites receive text input from users by inserting virtual keyboards into the VR scene for login, navigation, messaging, and more. A user utilizes the VR controller to point/click a certain key on the keyboard with the help of a ray from the controller object. When the text input is completed, the input is sent to the server by clicking the button that functions to transmit it, such as *Enter* or *Submit*.

According to our investigation, there is no standard specification for the virtual keyboard implementation in WebVR. As several JS libraries provide the key input functionalities [11, 12, 13, 14], WebVR developers can implement key input logic by embedding such JS libraries on their WebVR site. Keyboard libraries have minor differences in the shapes and layouts of keyboards (see Figure 2), but share the core functionalities: receiving and transmitting input values.

In this paper, we assume that the target VR site uses the *aframe-keyboard* (see Figure 2a), which provides intuitive features that are easy to deploy in the A-Frame WebVR environment. The aframe-keyboard has a typical keyboard shape; its key layouts are similar to those of physical keyboards, consisting of 46 keys, including a *Submit key* that triggers transmitting the

(a) aframe-keyboard (b) three-mesh-ui

(c) aframe-super-keyboard (d) vr-keyboard

Figure 2: The WebVR virtual keyboards provided by the JS libraries

input values to the server. Virtual keyboards are generally represented as 2D objects. Since the rotation of this entity makes it difficult for users to enter keys, we assumed that the keyboard entity does not rotate in the VR scene. The attacker can have prior knowledge of the keyboard, such as the layout, which is publicly available online.

3. Problem

This section presents the threat model of keylogging based on the vulnerability of VR controllers, as well as challenges to enabling an effective keylogging attack.

3.1. Vulnerability to VR Controllers

A VR controller can be used in WebVR applications via the *Gamepad* API [15], which allows to read the pose and orientation, as well as actuate haptic features. Through this API, a website can keep track of any VR input from buttons and thumbsticks. Since the Gamepad API handles such sensitive information, it is available only in a secure context (i.e., HTTPS) to prevent eavesdropping.

Despite the VR controller being maintained in a secure context, we discover an unusual circumstance where it can be accessible from malicious sites. Figure 1 shows that a user opens two different sites, WebVR and attacker

pages, on the left and right sides, respectively, on the same browser. Surprisingly, the attacker site can access the Gamepad API used in the WebVR site through the navigator.getGamepads() query, which allows the attacker site to obtain the main object (i.e., the *Gamepad* object) in the API. This entails a critical vulnerability that allows the attacker site to collect all the inputs from the WebVR site as long as both sites run at the same time.

Such a vulnerability is caused by the limitation that the Gamepad API does not provide any access control mechanism. Fortunately, the HMD, one of the main VR devices, does not suffer from this vulnerability due to its mode management feature. The *VRDisplays* object, corresponding to HMD control, maintains a flag indicating whether a webVR site runs in the VR mode while using the HMD to display VR scenes. Once the flag is set, other sites cannot access the HMD at the same time.

3.2. Threat model

We then define a threat model to be detailed as follows. We introduce a *WebVR attacker* to infer user keystrokes in WebVR. A WebVR attacker is a *classic web attacker* [16]. The attacker controls his/her own website and entices users to visit the malicious website. We have slightly expanded the web attacker model by assuming that the attacker site is open while the user enters virtual keyboard inputs on the target WebVR site. It is often encountered during web browsing. For example, a number of websites could be opened at the same time within multiple browser windows, and multiple webpages from different origins could be loaded into the iframe elements [17] embedded in a single webpage. In this situation, the attacker can obtain the Gamepad object by calling the navigator.getGamepads() API. It is worth noting that the attacker does not have to be a WebVR site; it can conduct the keystroke inference attack without any VR-related permissions. Furthermore, obtaining the Gamepad object does not require any modification to the target site or additional app installation. We assume that the attacker in this threat model has no limitations on analyzing the target keyboard in advance as well as abusing collected data.

3.3. Key logging attack and challenges

Although the attacker can collect all the inputs made on the target site, it is not possible for the attacker to be aware of what text the user entered. This is because the attacker only gathers raw input data such as the controller's pose/orientation and button-pressing status over time. In order to

infer the keystroke in WebVR while fully exploiting the vulnerability, we should address the following challenges:

- **C1. key- and app-clicks classification:** The attacker can eavesdrop on the VR controller when a user makes clicks. However, app- and key-clicks are not explicitly distinguished in the raw input data, so the attacker does not know which click is a key-click.
- **C2. Individual key recognition:** Virtual keyboards can be placed anywhere in the 3D space of a VR scene. However, it is impossible for the attacker to acquire the position from which the keyboard renders on the target site. The attacker does not know which specific key is pressed by a user.

4. VRKeyLogger Attack

4.1. Overview

We design VRKeyLogger, a VR keystroke inferencing framework, that supports a high degree of 1) accuracy and 2) robustness even if the attacker does not know the time of key input and where the virtual keyboard is located in a 3D space. The key enabler of VRKeyLogger is leveraging the absence of controller device isolation among the cross-origin websites. We first study the user-specific characteristics of keystrokes and collect information through the GamePad API. We then use an SVM-based classification algorithm to classify key-clicks among all user inputs (described in § 4.2). Moreover, we design a key-click identifier to infer exact keystrokes by transforming the hit-point coordinates of key-clicks into the local coordinate system known to attackers (described in § 4.3 and § 4.4).

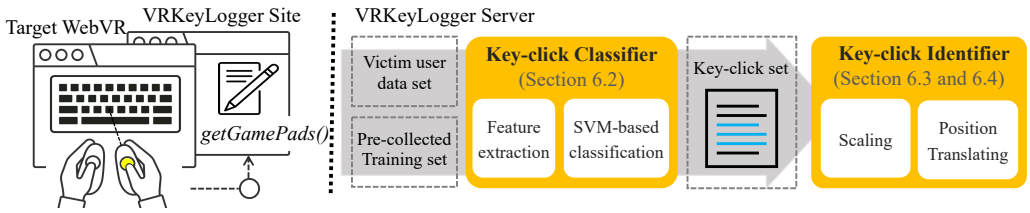


Figure 3: The overall procedure of VRKeyLogger attack

Figure 3 illustrates how VRKeyLogger infers keystrokes from the virtual keyboard through the use of controller gestures. The overall procedure of VRKeyLogger consists of two parts:

- 1) VRKeyLogger site: The VRKeyLogger site is loaded into a different window (or an iframe) while a user visits the target WebVR site. It continuously collects the controllers' sensor data provided by the GamePad API and sends it to the VRKeyLogger server.
- 2) VRKeyLogger server: Upon receiving data, the VRKeyLogger server extracts key features which are clearly distinguished from app-clicks and then classifies key-clicks with our SVM-based classifier. (§ 4.2). It then identifies which key is clicked by transforming hit-point coordinates observed on the VRKeyLogger site to the local coordinate system of the target virtual keyboard (§ 4.3 and § 4.4).

4.2. Key-click Classification Model

Each WebVR site has a different key input time, for which an attacker needs to know the exact time in order to infer keystrokes. The GamePad API provides the real-time position $\boldsymbol{P_i}$ and orientation $\boldsymbol{O_i}$ of the VR controller displayed in the 3D scene, where i denotes the time instant. $\boldsymbol{P_i}$ is defined as a 3D vector, $\langle p_{i,x}, p_{i,y}, p_{i,z} \rangle$, where $p_{i,x}$, $p_{i,y}$, and $p_{i,z}$ denote x-, y-, and z-axes coordinates, respectively. And, $\boldsymbol{O_i}$ is defined as another 3D vector, $\langle o_{i,x}, o_{i,y}, o_{i,z} \rangle$, where $o_{i,x}$, $o_{i,y}$, and $o_{i,z}$ denote angles of rotation about the x-, y-, and z-axes, respectively. Note that we only focus on $\boldsymbol{P_i}$ and $\boldsymbol{O_i}$ at the point where the click event happens. We empirically found that it is possible to effectively classify key-clicks based on such click event data only. More specifically, a click event is divided into two parts: the *down-click* event when pressing a button and the *up-click* event when releasing the button. As a user may move the controller between two events, we collect the data of all down- (i.e., $\boldsymbol{P_i^d} \in \boldsymbol{P^d}$ and $\boldsymbol{O_i^d} \in \boldsymbol{O^d}$) and up-click (i.e., $\boldsymbol{P_i^u} \in \boldsymbol{P^u}$ and $\boldsymbol{O_i^u} \in \boldsymbol{O^u}$) events while the target WebVR is active.

However, it is difficult to identify which direction the user is pointing only with this limited information. To address this, we additionally collect hit-points $\boldsymbol{H_i}$ corresponding to each click event. We place a canvas element on the VRKeyLogger site that renders a large 2D plane entity in front of the controllers. It also renders the VR controllers with rays, which automatically share the GamePad objects with the target VR site. We then collect the hit-point $\boldsymbol{H_i}$, defined as the point at which the controller's rays collide with the

plane entity. $\boldsymbol{H_i}$ is defined as a 2D vector, ($\langle h_{i,x}, h_{i,y} \rangle$), where $h_{i,x}$ and $h_{i,y}$ respectively denote x- and y- axes coordinates in the local coordinate system of the plane. Note that we collect the hit points for down- and up-click events ($\boldsymbol{H_i^d} \in \boldsymbol{H^d}$ and $\boldsymbol{H_i^u} \in \boldsymbol{H^u}$). Although the hit-point does not tell an attacker which objects are clicked by the user, it would be a great hint to infer the keystroke as it contains the click coordinate which is projected into the 2D space of the attacker.

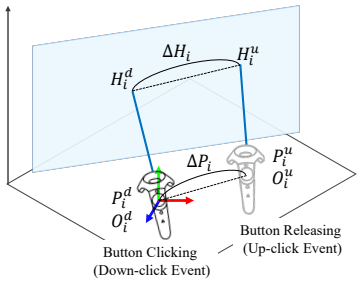


Figure 4: Notations description

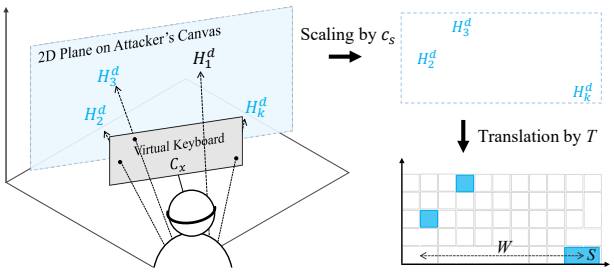


Figure 5: Procedures of key-click identification

The VRKeyLogger site collects the data set $\{\boldsymbol{P^d}, \boldsymbol{P^u}, \boldsymbol{O^d}, \boldsymbol{O^u}, \boldsymbol{H^d}, \boldsymbol{H^u}\}$ for all down- and up-click events made by the target WebVR site; it then sends the data set to the VRKeyLogger server after the target WebVR site finishes. For each click event, the VRKeyLogger server calculates the distance of controller positions $\Delta \boldsymbol{P_i} = \boldsymbol{P_i^u} - \boldsymbol{P_i^d}$ and hit-points $\Delta \boldsymbol{H_i} = \boldsymbol{H_i^u} - \boldsymbol{H_i^d}$ between the down- and up-click events, by the Euclidean distance formula. As a result, the tuple $\{\boldsymbol{P_i^d}, \boldsymbol{P_i^u}, \boldsymbol{O_i^d}, \boldsymbol{O_i^u}, \boldsymbol{H_i^d}, \boldsymbol{H_i^u}, \Delta \boldsymbol{P_i}, \Delta \boldsymbol{H_i}\}$ shown in Figure 4 is spawned for a single click. Note that we exclude $\boldsymbol{P_i^d}$ and $\boldsymbol{P_i^u}$ from the SVM feature set since they have similar values to app-clicks which lower the classification accuracy. All the features used in the key-click classification model are summarized in Table 1. Finally, the classifier returns a *key-click set* consisting of key clicks only.

It is worth noting that the VRKeyLogger site can record the timestamp of each click along with the data described above. This allows an attacker to extract the time interval between consecutive clicks as well as the click duration for down- and up-click events. Although they seem to be a great feature for the classification, time-related information widely varies for individual users (e.g., type quickly or slowly) and often shows values similar to app-clicks. For this reason, we do not include the time-related information in the classifier feature; instead, we use it after the classification to improve

Table 1: An instance of key-click classification model for a single click

Feature	Element	Description
$\boldsymbol{O_i^d}$	$\langle o_{i,x}^d, o_{i,y}^d, o_{i,z}^d \rangle$	A controller orientation at the down-click event
$\boldsymbol{O_i^u}$	$\langle o_{i,x}^u, o_{i,y}^u, o_{i,z}^u \rangle$	A controller orientation at the up-click event
$\boldsymbol{H_i^d}$	$\langle h_{i,x}^d, h_{i,y}^d \rangle$	A hit-point at the down-click event
$\boldsymbol{H_i^u}$	$\langle h_{i,x}^u, h_{i,y}^u \rangle$	A hit-point at the up-click event
$\Delta \boldsymbol{H_i}$		A distance between down-click and up-click hit-point
$\Delta \boldsymbol{P_i}$		A distance between down-click and up-click controller position

the accuracy. A detailed explanation is covered in Section 6.2.

4.3. Key-click Identification

The WebVR hosts can randomly set the scale and position of a virtual keyboard, which an attacker must know to infer the keystrokes. To uncover the hosts' keyboard settings, we introduce a *key-click identifier*. Figure 5 shows the overall process of key-click identification. The basic idea of this algorithm is to use the local coordinate system that represents the inherent coordinate system of the target virtual keyboard, regardless of its location within a VR scene. Thus, an attacker investigates the local coordinates of the target keyboard in advance and transforms the hit-points of the key-click set into them.

We derive the coordinate transform as follows:

$$\boldsymbol{H_i'} = c_s \cdot \boldsymbol{H_i^d} + \boldsymbol{T} \tag{1}$$

where c_s denotes the scaling factor, $\boldsymbol{H_i'}$ and $\boldsymbol{T}$ denote 2D vectors of the transformed coordinates ($\langle h_{i,x}', h_{i,y}' \rangle$) and the translation vector ($\langle t_x, t_y \rangle$), respectively. As the transformed hit-point $\boldsymbol{H_i'}$ is a specific point in the local coordinate system of the target keyboard, it directly corresponds to a certain key on the keyboard. It is worth noting that if the hit-points of the down- and the up-click events are not placed on the same key, the key is not entered correctly, which is likely to be filtered out by our key-click classification. Therefore, we only consider the hit-point of the down-click event to infer the keystroke.

Scaling factor c_s. The scaling factor c_s is a coefficient to adjust the position of the key-clicks in proportion to the size of the target keyboard. To correctly determine this factor, it is necessary to know the size of the target

keyboard. However, it is impossible for attackers to get this sensitive information. Therefore, an attacker has to infer the keyboard size based on at least two points in the local coordinate system, which correspond to certain points on the target keyboard. To this end, we assume that the target keyboard is rendered at the center of the user, while the y- and z-axes positions, as well as the size of the keyboard are still unknown to the attacker. We also take advantage of the fact that users typically send the input value by pressing the *Submit* key at the end of keystrokes. As VRKeyLogger can classify key-clicks and app-clicks, VRKeyLogger can distinguish the last key-click instance (i.e., the k-th instance) among consecutive key-clicks as the Submit key. Therefore, the scale factor c_s can be defined as follows:

$$c_s = \frac{W}{(h^d_{k,x} - C_x), \cdot 2} \tag{2}$$

where W denotes the keyboard width in the local coordinate system, $h^d_{k,x}$ is the x-axis position of the hit-point of the Submit key, and C_x denotes the x-axis position of the user's head.

Translation vector $\boldsymbol{T}$. The translation vector $\boldsymbol{T}$ moves the scaled coordinates, derived by multiplying the hit-point $\boldsymbol{H^d_i}$ by c_s, to the local coordinate system of the target keyboard, which is proactively known to the attacker. $\boldsymbol{T}$ can be easily derived by calculating the distance between the position of the Submit key in the local coordinate system (i.e., $\boldsymbol{S}$) and the scaled position of the Submit key-click (i.e., $c_s \cdot \boldsymbol{H^d_k}$), as follows:

$$\boldsymbol{T} = \boldsymbol{S} - c_s \cdot \boldsymbol{H^d_k} \tag{3}$$

Now, we can obtain the transformed coordinates $\boldsymbol{H'_i}$ through Equation (1), based on c_s and $\boldsymbol{T}$, that directly correspond to a certain key of the target keyboard.

This key-click identification has the advantage of not requiring awareness of the hand that holds the controller, because it exploits hit-point positions rather than the position and orientation of the controller. On the other hand, the key identification inference accuracy is affected by the observed hit-point position of the Submit key. In general, in order to click a key, a user only needs to click a point within the rectangular area of the key element. If the user clicks on a point far from the center of the area, it is difficult to identify the key value that is transformed based on this point, resulting in poor attack performance. In the next section, we introduce a technique to improve the accuracy of the key-click identifier.

4.4. Improvement Technique for Key-click Identification

The problem described above occurs because all hit-points are transformed by Equation (1) derived by only a single Submit key. We can alleviate this limitation by employing the most clicked key as the reference key to derive the equation. We first group the hit-point coordinates with small differences to find the most clicked key *ref_key*. For example, if a user has entered the backspace key the most, the ref_key will be the backspace key. VRKeyLogger can determine the scaling factor c_s and $\boldsymbol{T}$ based on the mean of hit-points of the ref_key and the local coordinate of the backspace key instead of $\boldsymbol{H_k^d}$ and $\boldsymbol{S}$, respectively.

It is also possible to further improve in the case that an attacker is aware of some successive strings (e.g., HTTPS://, user ID, name, phone number, and so on) included in the key-click set. For example, since the URL address always includes a protocol scheme (e.g., HTTP) and generic TLDs (e.g., com or net), it is easy to grasp partial strings in the context of site navigation. In this case, an attacker can add a known string to find a more accurate click position. With this, it is possible to increase the accuracy of keystroke inference even with a small number of key-clicks.

5. Implementation

The implementation of VRKeyLogger mainly consists of the VRKeyLogger site and the VRKeyLogger server. We embedded the VRKeyLogger site in the testing WebVR sites via an iframe element and collected the VR controller data using the GamePad API. The VRKeyLogger site loads the *A-Frame* JS library to obtain hit-points. It sends the collected data to the VRKeyLogger server using WebSocket [18] at the time when the hosting site closes (captured by window.onbeforeunload). To collect the ground truth of each key-click, we slightly modified the testing WebVR sites to record labels, which contain 1 and the corresponding key value for a key-click otherwise 0, for each click event. Afterward, we added labels to the test data collected from the VRKeyLogger site to estimate the accuracy. The entire implementation, including the key-click classification and identification, is about 645 lines of code, which is lightweight and easy to deploy.

6. Evaluation

In this section, we evaluate VRKeyLogger by answering the following questions:

1. How accurately can VRKeyLogger classify clicks on key inputs?

2. How accurately can VRKeyLogger identify the pressed key without knowing the location of the virtual keyboard on the target WebVR site?

3. How much performance overhead has occurred in deploying VRKeyLogger?

To measure the efficacy of the presented attack, we recruited nine participants and investigated their responses to the four attack scenarios. Section 6.1 describes the design of the user study. The results of the attack accuracy are described in Sections 6.2 and 6.3. We also evaluate the performance overhead of VRKeyLogger in Section 6.4.

6.1. Experimental Design

To investigate the user impact of the attack, we recruited a total of nine participants, consisting of six males and three females (average age is 25.6). We prepared a spacious room to play WebVR with an HTC VIVE pro [19] on a Windows 10 host. Participants were asked to carry out WebVR assignments and a survey according to the instructions. The total experimental time took about 30 minutes. All participants have VR experience, two of whom have experienced VR more than 10 times, and seven of them are right-handed.

To evaluate the key-click classification accuracy, we prepared three open-source VR apps: A-Blast [20], Moon-Rider [21], and A-Painter [22], each shows differences in classification difficulty due to the use of controllers. A-Blast [20] is a shooting game in which players shoot flying monsters with blaster guns. Moon-Rider [21] is a rhythm game, players crush the beats that get closer along the lane with their two fists. A-Painter [22] is a 3D painting utility app that allows users to paint in a 3D space with brushes. In all three apps, user interactions are manipulated by VR controllers.

We slightly modified these WebVR sites to insert an aframe-keyboard and received the user's keyboard input once during the entire play. To make a difference at the time of key input, we inserted the keyboard input sequence for A-Blast and Moon-Rider before the game started, while the A-Painter receives user input when the participants press the save button after finishing their drawing. Participants were instructed to enter the given characters (in this setting, we use "misys.430") and also 8-12 random characters.

To further analyze the performance of the key-click identification, we measured the identification accuracy by varying the scale and position of the virtual keyboard. We implemented an empty WebVR site that renders a virtual keyboard in front and instructed participants to click all the keys on the keyboard using a controller in either hand. Keyboard sizes were investigated for small (scale = 2, S for short), medium (scale = 3, M for short), and large (scale = 4, L for short), located at $\langle 0.48, 1.2, -2\rangle$, $\langle -0.72, 1.2, -2\rangle$, and $\langle -0.96, 1.2, -2\rangle$, respectively. In a survey that asked about the difficulty of entering keys by size, two of the participants answered that it was difficult to click on the S-sized keyboard, and seven answered that there was no difficulty in all clicks.

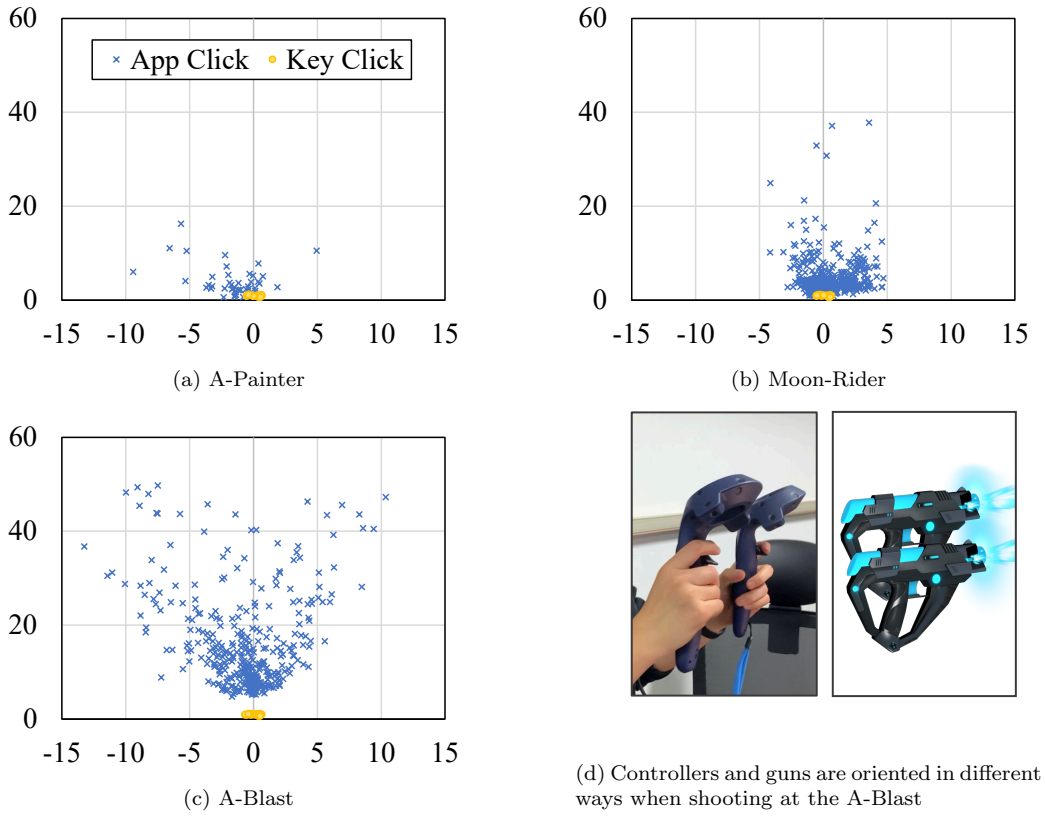
6.2. Key-click Classification Accuracy

For the classifier training, we collected click data (a total of 984 records) generated by one of the authors from all experimental VR sites. We used the SVM, particularly with the C-SVC multi-class categorization scheme and a poly-type kernel function. Classification accuracy is derived as the number of correctly classified click instances (the sum of true-positives and true-negatives) over the total number of click instances.

Table 2: Average key-click classification accuracy over three different WebVR sites

	A-Blast	Moon-Rider	A-Painter
Classification Accuracy (%)	99.55	90.93	91.46

Figure 6 shows the coordinates of hit-points for three testing WebVR sites. As similar patterns were observed for all participants, in Figure 6, we only present the result of Participant 1 as a representative. Overall classification accuracy is summarized in Table 2. It is worth noting that we intentionally placed the virtual keyboard in a certain position, which makes the hit-points of key-clicks as similar as possible to those of app-clicks, thereby excluding the factors that facilitate classification. Nevertheless, the app-clicks of shooting enemies in A-Blast are distinct from the key-clicks (Figure 6c) because the controllers are directed upwards when shooting (Figure 6d). As a result, the classification accuracy of A-Blast is as high as 99.55%. On the other hand, because A-Painter can utilize the 3D space widely, the controller usage patterns



(a) A-Painter

(b) Moon-Rider

(c) A-Blast

(d) Controllers and guns are oriented in different ways when shooting at the A-Blast

Figure 6: Hit-point coordinates of three test WebVR sites collected from u1 and the appearance of playing A-Blast

are widely varied, user-by-user. Due to this effect, the classification accuracy of A-Painter is 91.46%, which is less than A-Blast. Moon-Rider showed the lowest accuracy of 90.93%, indicating that there are many app-clicks similar to key-clicks. In particular, at the beginning stage that selects songs to play, UI buttons are placed very similar to keyboard buttons, resulting in misclassifications.

However, attackers can exploit additional time information to improve the classification accuracy. For example, since the text input consists of multiple keys rather than a single key, the attacker can exclude a non-continuous click even though it is classified as a key-click (e.g., Line 15 in Table 3). In addition, a Submit key normally entails a time delay (e.g., data transmission or navigation) until the next click. Therefore, the attackers can easily specify a certain key that is most likely to be a Submit key, for instance, Line 8 in

Table 3: An example of a key-click classification result for Moon-Rider. It indicates that the 8th row is the most likely Submit key due to the down-click time interval.

	down-click time	orientation.x	orientation.y	orientation.z	...	pos_dist	hit-point_dist	predict
1	36874	4.819653016	11.8246399	9.080704258	...	0.00221792	0.010142182	1
2	37256	3.875373176	10.51913586	11.21453805	...	0.001244302	0.008786598	1
3	37707	3.979222059	7.40985049	11.56059523	...	0.00222138	0.03670739	1
4	38140	4.008094196	4.525896043	12.26025726	...	0.003687316	0.040945209	1
5	39140	6.974631091	6.50873339	11.14097035	...	0.001674669	0.012192948	1
6	40740	12.18328019	6.841808111	8.831105565	...	0.002964103	0.012435971	1
7	41190	11.63425033	4.196085116	9.953865428	...	0.001782217	0.010027515	1
8	42923	2.556396856	-5.406556759	13.52199273	...	0.002083228	0.023135344	1
9	48306	16.01305734	-1.488898826	4.250287424	...	0.00866908	0.039959986	0
10	49056	8.996248618	1.569822596	2.372020145	...	0.012435546	0.122233245	1
11	49490	9.263585209	-6.720666966	7.336628715	...	0.005423505	0.056457468	1
12	49923	5.796454495	2.701196264	1.598850448	...	0.010079301	0.047963183	1
13	50190	20.96905253	3.212126531	8.91651863	...	0.019800085	0.120838017	0
14	50423	4.006456725	4.019317717	1.828322099	...	0.03365967	0.079035807	0
15	51023	5.021948977	-3.479168722	6.795713714	...	0.019338071	0.067325333	1
16	51540	21.24273408	4.349179489	7.91355118	...	0.008074717	0.045642979	0

Table 3, which has the longest down-click time interval. In this way, attackers can further remove false-positive labels, drastically reducing the search space.

6.3. Key-click Identification Accuracy

Table 4: Average key-click identification accuracy over three different sizes of virtual keyboards

	Small	Medium	Large
Inference Accuracy (%)	99.10	95.50	91.47

To understand the attack performance of VRKeyLogger, we evaluate how accurately the key-click identifier determines the input value. We verified the performance of the key-click identifier by the number of correctly identified instances over the total number of key-clicks. Note that the key-click identifier applies the accuracy improvement technique based on the *ref_key* as presented in Section 4.4. Table 4 shows the key-click identification accuracy of nine participants when the sizes of virtual keyboards vary in each S, M, and L. The result shows that the user's key input can be identified with a high success rate. It also shows that the identification accuracy tends to slightly decrease in L. This is because the larger key size makes for a wider

valid click range, leading to more errors when transforming the hit-point coordinates. However, the L-sized keyboard used in the experiment is rarely used in practice. According to the survey, 78% of participants answered that there was no problem clicking on the M keyboard, indicating the result is sufficiently practical.

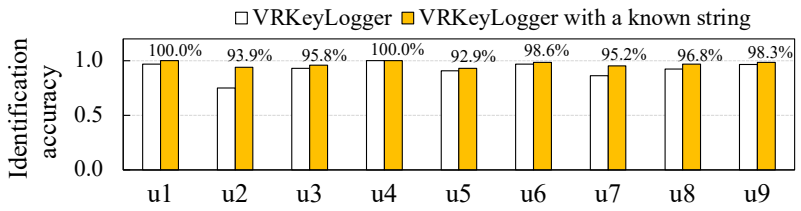


Figure 7: Average key-click identification accuracy per user

In addition, Figure 7 shows the key-click identification accuracy for each user in three real-world WebVR sites. The x- and y-axes represent the user ID and the average identification accuracy of three VR sites, respectively. The white and yellow bars present the result, with and without the accuracy improvement technique with a known string (see the second paragraph in Section 4.4), respectively. Note that when the input keystroke contains a certain text sequence that is known to VRKeyLogger (i.e., "misys.430" in our user study), the key identifier yields a higher accuracy. As shown in the figure, our attack shows robust accuracy regardless of users. In addition, our optimization technique is highly effective as the lowest accuracy of VRKeyLogger with the known string (i.e., the yellow bar) is 93.9% while that of VRKeyLogger (i.e., the white bar) is 74.9%. Consequently, VRKeyLogger results in an identification accuracy of 92% on average for all users, and the accuracy is improved up to 96.8% when applying VRKeyLogger with the known string.

The result shows that our technique outperforms the previous results presented in [2, 3], which provide keystroke inference success rates of 58% and 69.7%, respectively. This is because our approach captures clearer hit-point information for keystroke inference than side information such as the user's motions or WiFi signals. This figure also demonstrates that participants who clicked the controller with both left and right hands (u3, u5, u7, and u9 in Figure 7) did not affect the attack performance results, indicating that our attack is robust.

6.4. Performance Overhead

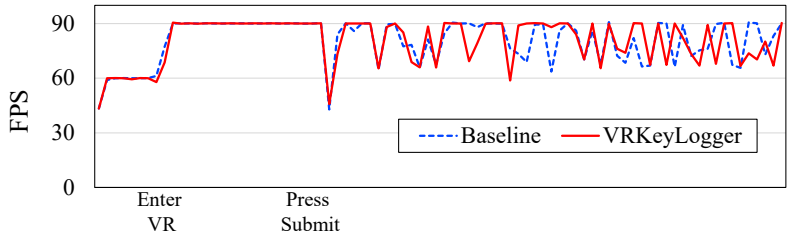


Figure 8: FPS variations in baseline and VRKeyLogger while Moon-Rider is running

To assess the overall performance overhead during the experience of WebVR, we measured the FPS changes while running Moon-Rider, which is the most complex test site. High overhead may increase the likelihood that victims detect abnormalities such as keystroke hijacking, as well as an abrupt FPS reduction decreases the user's sense of immersion in the VR mode and may cause a poor user experience [23]. Figure 8 shows the result. Note that the Firefox browser caps the FPS at 60 in the non-VR mode and relaxes it up to 90 in the VR mode. Compared to the baseline in which the attack was not conducted, the FPS reduction of VRKeyLogger is negligible. In addition, all participants answered in the survey that they were not aware of the attack during the experiment. Thus, VRKeyLogger does not affect the WebVR performance.

7. Discussion

7.1. Applicability to WebXR

To support the increasing demand for immersive computing, WebXR [24], which offers an integrated platform to support VR and AR (Augmented Reality) devices, has been introduced on the Web. Although many WebXR features and specifications are still under development, commercial web browsers, such as Chrome and Edge, partially support WebXR. Fortunately, WebXR does not have the vulnerability of the Gamepad API, as it accesses the VR controllers through a new interface, *xrSession*. It is impossible to access the xrSession used by others. The request for a new xrSession is pending until the existing xrSession finishes; therefore, none of the XR interfaces can be called at the same time.

However, WebXR still has a vulnerability that can be exploited by VRKeyLogger. Lee et al. introduced a WebXR attacker threat model for WebXR environments [25] that is also applicable to our keystroke inference attack. A WebXR attacker is a third-party library that provides 3D objects (e.g., advertising) to the hosting VR scene. In this case, as of now, WebXR does not support the origin separation mechanism that allows third-party JS scripts to securely render entities within the hosting VR scene. The WebXR publisher has no other option than to run the third-party JS scripts within the host's execution context to render the resources of the third-party service providers. Consequently, an abusive third-party library can access the VR controller objects created by the host site and collect information to infer the user's keystrokes by way of VRKeyLogger.

7.2. Mitigation

Now we suggest possible approaches to mitigate the virtual keyboard inference threat. One approach is to adopt a secure keyboard that is capable of changing the key arrangement for private inputs (e.g., passwords). Similarly, virtual keyboards with deformed 3D shapes rather than 2D shapes (e.g., a 2D square) can help to mitigate the VR key inference attack. For example, when applying a curved-shaped keyboard, it is not easy to collect hit-points by simply placing the plane entity in front, and additional information such as curvature is required, making it much more difficult to conduct the attack. In addition, techniques for enhancing authentication by combining various pieces of information can be effective in mitigating VR side-channel attacks. For example, existing studies have shown that the combination of users' biometric information (e.g., human visual system [26], skull [27]) or several wearable computing devices (e.g., smartwatches [28], wristbands [29]) can enhance the security of VR authentication mechanisms.

8. Related Work

Keystroke inference on a physical keyboard. There have been several attempts to infer a user's inputs on a physical keyboard. Liu et al. [30] introduced an acoustic-based keystroke inference attack on a physical keyboard. In this attack, an attacker collects keystroke sounds using a smartphone placed close to a keyboard while a victim inputs sensitive information on the keyboard. However, it does not apply to VR environments. This is primarily owing to the absence of physical interactions with a keyboard. For

example, in VR environments, no sound is generated during typing inputs through virtual keyboards. Other studies [31, 32] showed the feasibility of leveraging a smartwatch to infer keystrokes on a physical keyboard. They have based on observations that the smartwatch's movement varies depending on the keystroke location. However, they suffer from low accuracy because the smartwatch can capture the movement of only the hand where the watch is worn.

Keystroke inference on a touch screen. Many studies have raised new security concerns by inferring a user's tap inputs, called tapstrokes, on an on-screen keyboard of a smartphone. Towards this, they leveraged sensors present in a smartphone. For example, some of them [33, 34, 35] used built-in microphones to collect sounds, generated due to tapstrokes. Other works [36, 37, 38, 39] have captured a smartphone's movement, caused by tapstrokes, with the use of IMU sensors, including accelerometers and gyroscopes. These works then exploited the sounds and movements as a side-channel for inferring tapstrokes. That is, they rely on the phenomena caused by physical interactions between a user and a keyboard, which are not common in VR environments. Another class of works [40, 41] utilized a front camera of a smartphone to track a victim user's eye movement patterns during input of sensitive information. We believe that our proposed approach would be incorporated with these vision-based techniques to further improve inference accuracy.

Keystroke inference in a VR environment. Unlike in the physical environment, there have been few studies of keystroke inference attacks in the VR environment. Ling et al. [2] introduced a vision-based side-channel approach that keeps track of the VR headset movements with a stereo camera. This study presented a preliminary approach that requires malicious app installation on a user's computer, resulting in a low inference success rate of 58%. VR-Spy [3] is another virtual keystroke inference technique exploiting the Channel State Information (CSI) of WiFi signals. Assuming an attacker has access to the user's WiFi device, VR-Spy infers keystrokes by extracting a unique pattern of the captured CSI waveform from the VR controller gestures associated with each virtual key input. Both studies differ from our attack in that they strongly assume that the size and location of the virtual keyboard are always fixed and the attacker always knows when the user types the keyboard.

VR authentication techniques. As a secure user interaction is important for VR users, a number of studies have proposed VR-specific secure au-

thentication techniques which are safe from the keystroke inference attack. LookUnlock [42] is an HMD authentication technique that uses passwords configured to enter virtual objects in the correct order. Oculock [26] is a biometric authentication system designed for VR platforms, utilizing human visual systems such as eye glove movements, eyelids, and extraocular muscles triggered by immersive content. RubikBiom [43] showed that hand movement patterns with a knowledge-based authentication scheme (e.g., PIN) can be leveraged to establish an additional security layer. As such, VR authentication techniques that combine additional information can hide the movement of VR users and consequently help defend against side-channel attacks from bystanders. However, as such approaches only focus on the authentication, general keyboard inputs, except the authentication, are still exposed to the keystroke inference threat.

9. Conclusion

In this paper, we devised a new side-channel attack that infers keystrokes of the virtual keyboard entered by VR controllers while the user visits a WebVR site. We first discovered the security flaw in the WebVR GamePad API that allows web attackers to capture the VR controller usage patterns. We developed a key-click classification that extracts clicks on the virtual keyboard among all clicks that occur during app use. We also introduced a key-click identification that transforms hit-point coordinates into the local coordinate system known to attackers, establishing a practical keystroke inference framework. Our evaluation, conducted with nine participants, shows that our approach outperforms the previously reported results, as VRKeyLogger can successfully infer user inputs with 96.8% of keystroke inference accuracy with negligible overhead. Thus, we demonstrate that an attacker can effectively identify keystrokes entered into WebVR sites and remark the importance of pursuing safe authentication techniques in WebVR.

Acknowledgements

This work was supported in part by NRF-2020R1F1A1076425 and NRF-2021R1A4A1032252.

References

[1] W3C, WebVR 1.1, https://immersive-web.github.io/webvr/spec/1.1/, last visited: 2022-03-05.

[2] Z. Ling, Z. Li, C. Chen, J. Luo, W. Yu, X. Fu, I Know What You Enter on Gear VR, in: 7th IEEE Conference on Communications and Network Security, CNS, IEEE, 2019.

[3] A. A. Arafat, Z. Guo, A. Awad, VR-Spy: A Side-Channel Attack on Virtual Key-Logging in VR Headsets, in: IEEE Virtual Reality and 3D User Interfaces, VR, IEEE, 2021.

[4] A-Frame, A WebVR Implementation Platform, https://aframe.io/docs/0.9.0/introduction/, last visited: 2022-03-05.

[5] L. OpenGL, Coordinate Systems, https://learnopengl.com/Getting-started/Coordinate-Systems, last visited: 2022-03-05.

[6] Three.js, A JavaScript 3D Library, https://threejs.org/, last visited: 2022-03-05.

[7] M. D. Network, Document Object Model (DOM), https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction, last visited: 2022-03-05.

[8] M. D. Network, Inputs and input sources, https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API/Inputs, last visited: 2022-03-05.

[9] A-Frame, A-Frame Raycaster System, https://aframe.io/docs/1.3.0/components/raycaster.html, last visited: 2022-03-05.

[10] Three.js, Three.js Raycaster System, https://threejs.org/docs/index.html#api/en/core/Raycaster, last visited: 2022-03-05.

[11] GitHub, aframe-keyboard, https://github.com/WandererOU/aframe-keyboard, last visited: 2022-03-05.

[12] GitHub, three-mesh-ui, https://github.com/felixmariotto/three-mesh-ui, last visited: 2022-03-05.

[13] GitHub, aframe-super-keyboard, https://github.com/supermedium/aframe-super-keyboard, last visited: 2022-03-05.

[14] GitHub, vr-keyboard, https://github.com/erosmarcon/vr-keyboard, last visited: 2022-03-05.

[15] M. D. Network, GamePad API, https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API, last visited: 2022-03-05.

[16] A. Barth, C. Jackson, J. C. Mitchell, Securing Frame Communication in Browsers (2009).

[17] M. D. Network, iframe: Inline Frame Element, https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe, last visited: 2022-03-05.

[18] M. D. Network, Web Socket, https://developer.mozilla.org/en-US/docs/Web/API/WebSocket, last visited: 2022-03-05.

[19] HTC, VIVE Pro Specs, https://www.vive.com/us/product/vive-pro/, last visited: 2022-03-05.

[20] M. VR, A-Blast, https://aframe.io/a-blast/, last visited: 2022-03-05.

[21] Supermedium, Moon Rider, https://moonrider.xyz/, last visited: 2022-03-05.

[22] M. VR, A-Painter, https://aframe.io/a-painter/, last visited: 2022-03-05.

[23] J. E. Bos, W. Bles, E. L. Groen, A theory on visually induced motion sickness, Displays 29 (2) (2008) 47–57.

[24] W3C, WebXR Device API, https://www.w3.org/TR/webxr/, last visited: 2022-03-05.

[25] H. Lee, J. Lee, D. Kim, S. Jana, I. Shin, S. Son, AdCube: WebVR Ad Fraud and Practical Confinement of Third-Party Ads, in: 30th USENIX Security Symposium, USENIX Security, USENIX Association, 2021.

[26] S. Luo, A. Nguyen, C. Song, F. Lin, W. Xu, Z. Yan, OcuLock: Exploring Human Visual System for Authentication in Virtual Reality Head-mounted Display, in: 27th Annual Network and Distributed System Security Symposium, NDSS, The Internet Society, 2020.

[27] S. Schneegass, Y. Oualil, A. Bulling, SkullConduct: Biometric User Identification on Eyewear Computers Using Bone Conduction Through the Skull, in: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, CHI, ACM, 2016.

[28] A. Bianchi, I. Oakley, Wearable authentication: Trends and opportunities, it Inf. Technol. 58 (5) (2016) 255–262.

[29] R. Zhang, N. Zhang, C. Du, W. Lou, Y. T. Hou, Y. Kawamoto, AugAuth: Shoulder-surfing resistant authentication for augmented reality, in: IEEE International Conference on Communications, ICC, IEEE, 2017.

[30] J. Liu, Y. Wang, G. Kar, Y. Chen, J. Yang, M. Gruteser, Snooping Keystrokes with mm-level Audio Ranging on a Single Phone, in: Proc. ACM MobiCom, 2015.

[31] X. Liu, Z. Zhou, W. Diao, Z. Li, K. Zhang, When Good Becomes Evil : Keystroke Inference with Smartwatch, in: Proc. ACM CCS, 2015.

[32] H. Wang, T. T. Lai, R. R. Choudhury, MoLe: Motion Leaks through Smartwatch Sensors, in: Proc. ACM MobiCom, 2015.

[33] S. Narrain, A. Sanatinia, G. Noubir, Single-stroke Language-Agnostic Keylogging using Stereo-Microphones and Domain Specific Machine Learning Categories and Subject Descriptors, in: Proc. ACM WiSec, 2014.

[34] I. Shumailov, L. Simon, J. Yan, R. Anderson, Hearing your touch: A new acoustic side channel on smartphones, in: arXiv preprint arXiv:1903.11137, 2019.

[35] H. Kim, B. Joe, Y. Liu, TapSnoop: Leveraging Tap Sounds to Infer Tapstrokes on Touchscreen Devices, IEEE Access 8 (2020) 14737–14748.

[36] Z. Xu, K. Bai, S. Zhu, TapLogger: inferring user inputs on smartphone touchscreens using on-board motion sensors, in: Proc. ACM WiSec, 2011.

[37] E. Miluzzo, A. Varshavsky, S. Balakrishnan, R. R. Choudhury, Tapprints: your finger taps have fingerprints, in: Proc. ACM MobiSys, 2012.

[38] L. Cai, H. Chen, TouchLogger: inferring keystrokes on touch screen from smartphone motion, in: Proc. HotSec, 2011.

[39] D. Ping, X. Sun, B. Mao, TextLogger : inferring longer inputs on touch screen using motion sensors, in: Proc. ACM WiSec, 2015.

[40] Y. Chen, T. Li, R. Zhang, Y. Zhang, T. Hedgpeth, EyeTell: Video-Assisted Touchscreen Keystroke Inference from Eye Movements, in: Proc. of IEEE Symposium on Security and Privacy, 2018.

[41] Y. Wang, W. Cai, T. Gu, W. Shao, I. Khalil, X. Xu, GazeRevealer: Inferring Password Using Smartphone Front Camera, in: Proc. of ACM MobiQuitous, 2018.

[42] M. Funk, K. Marky, I. Mizutani, M. Kritzler, S. Mayer, F. 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, ACM, 2019.

[43] F. Mathis, H. I. Fawaz, M. Khamis, Knowledge-driven Biometric Authentication in Virtual Reality, in: Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, CHI, ACM, 2020.

Declaration of interests

☐The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☒The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Kilho Lee reports financial support was provided by National Research Foundation.