Towards Secure User Interaction in WebXR

Chandrika Mukherjee
Purdue University
West Lafayette, IN, USA
cmukherj@purdue.edu

Arjun Arunasalam
Purdue University
West Lafayette, IN, USA
aarunasa@purdue.edu

Habiba Farrukh
University of California, Irvine
Irvine, CA, USA
habibaf@uci.edu

Reham Mohamed Aburas
American University of Sharjah
Sharjah, UAE
raburas@aus.edu

Z. Berkay Celik
Purdue University
West Lafayette, IN, USA
zcelik@purdue.edu

Abstract

Advancements in the extended reality (XR) has resulted in the emergence of WebXR, an XR-open standard interface that enables users to access immersive virtual environments via a browser without additional software. Following this, diverse applications are being developed for WebXR ranging from gaming and shopping to medical and military use. However, recent research indicates that various UI properties in WebXR, such as synthetic input and same-space overlapping objects, can be exploited by adversaries to manipulate users into unintentional actions, especially in the advertising ecosystem. The consequences range from system malfunctions and user data loss to financial and reputational impacts on several involved ad-stakeholders.

In this paper, we present our experience in designing a log framework that captures granular user interactions in a 3D WebXR environment to evaluate the impact of UI manipulation attacks on users. Leveraging data collected in a previous study, we demonstrate the use cases of our log framework for XR platforms and developers. In addition, we discuss practical lessons learned, highlighting the logger's limitations and outlining future directions. Our work brings us closer to realizing security-focused logging in WebXR environments to protect users from various UI manipulation attacks, thereby helping them preserve their autonomy within these immersive platforms.

CCS Concepts

• **Security and privacy → Usability in security and privacy**; • **Human-centered computing → Activity centered design**.

Keywords

WebXR; Security & Privacy; User Interaction Logging

ACM Reference Format:
Chandrika Mukherjee, Arjun Arunasalam, Habiba Farrukh, Reham Mohamed Aburas, and Z. Berkay Celik. 2025. Towards Secure User Interaction in WebXR. In *The 3rd International Workshop on Human-Centered Sensing, Modeling, and Intelligent Systems (HumanSys '25), May 6–9, 2025, Irvine, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3722570.3726880

1 Introduction

The WebXR API allows users to engage with immersive environments directly through a WebXR-enabled browser. With new input methods such as controllers and gaze, along with sensor-enabled immersion, WebXR differs significantly from the standard web. These UI properties increase the realism of WebXR environment. For example, transparency can create visual effects such as depth perception, flowing water, and shattering glass. Similarly, multiple objects can be placed within the same space in WebXR to create complex virtual scenes and intricate object interactions (e.g., various objects representing marine life in an underwater scene). This presents new opportunities for developing immersive web applications across various domains, including gaming [6], shopping [12], medical [4], and military [10] training.

However, prior works [3, 7, 9] demonstrate that security-sensitive UI properties in WebXR enable adversaries, particularly in the advertising ecosystem, to manipulate users into unintended actions or facilitate dark patterns. In the standard web, ads are delivered using the `<iframe>` HTML element, which isolates execution across different origins, including the ad content. However, in WebXR, there is no iframe-like element, requiring developers to allocate a portion of the scene to ads, thereby reducing control over content from separate origins.

Exploiting this lack of isolation, entities in the ad ecosystem (developers, ad service providers, and advertisers) can leverage security-sensitive UI properties (e.g., transparency, synthetic input, blind spots, and clicks registered by the first clickable entity) to manipulate user interactions. These manipulations can have serious consequences for users, including forced engagement with ad content, generating revenue for adversaries, and even data theft or malware downloads. In addition, they can financially impact other entities in the ad ecosystem or damage their reputation.

In our recent work on WebXR UI manipulation attacks [9], we introduced five new attacks, identified nine from prior research [3, 7], and proposed a taxonomy categorizing these 14 attacks into four groups based on the primary objectives of malicious actors within the ad ecosystem. These are *Click Manipulation* (Ⓐ), *Peripheral Exploitation* (Ⓑ), *Functionality Disruption* (Ⓒ), and *UI-based Privacy Leakage* (Ⓓ). Attacks within Ⓐ[1] trick users into unintentional clicks on ads to generate fraudulent revenue. Attacks within Ⓑ exploit areas outside the focus of users to generate illicit impressions or clicks on ads. Attacks in Ⓒ intentionally disrupt the functionality

[1]We use these notations henceforth to refer to respective categories.

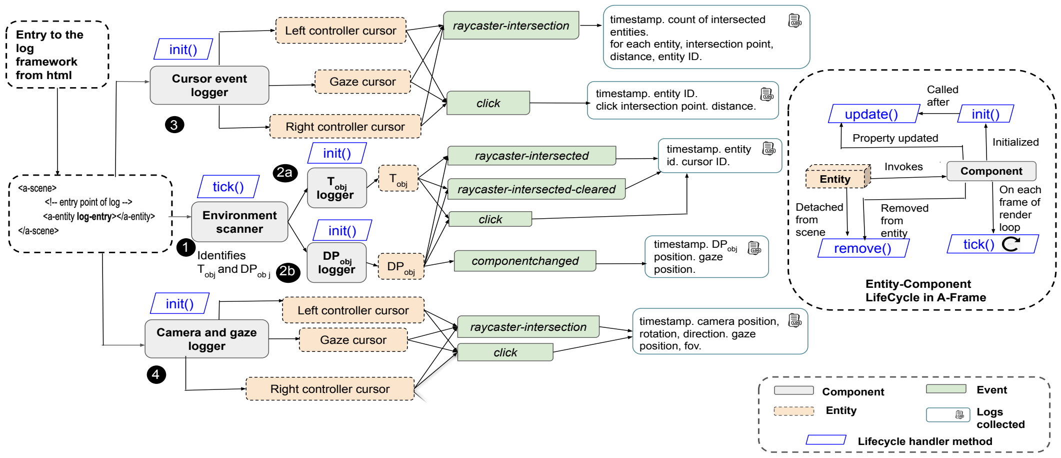


Figure 1: The design of our log framework.

of the app, preventing user-intended actions. Lastly, attacks in Ⓓ extract sensitive user information, compromising their privacy.

To analyze the impact of these attacks on users, we conducted a between-subjects user study with 100 users in our prior work [9]. We incorporated our identified 14 attacks within four different apps (gaming, reading, shopping, and travel) with varying interaction requirements. To capture granular 3D interaction data from the WebXR environment, we developed a log framework and integrated it with all 14 attacks in all four apps. This enabled us to analyze changes in user-interaction behavior in the presence of attacks. Additionally, it helped evaluate whether attack effectiveness varied based on the interaction requirements of each app.

In this paper, we present the design and implementation of the log framework for analyzing user interactions in WebXR, detailing its components and deployment. We demonstrate various use cases of the collected logs from our user study and explore the logger's functionality. For example, we analyze trends in user-intended and unintended interactions, attention patterns, and potential click confusion, and their implications. In addition, we discuss its limitations and outline directions for future work.

2 Logging Interactions in WebXR

Motivation. For our user study, we integrated each attack into four apps - gaming, reading, shopping, and travel, where users performed context-specific tasks such as reading a book, shopping for clothes, shooting targets, or exploring locations. We define task-related objects as T_{obj} and those potentially linked to dark patterns (e.g., ads) as DP_{obj}. To analyze user attention in a 3D scene over time or to determine whether clicks on DP_{obj} and/or T_{obj} resulted from genuine interest, it was necessary to develop a log framework.

Existing web analytics tools/libraries (e.g., Google Analytics [2], Firebase [5]) can automatically track events such as clicks, page navigation, and scrolling. Yet, in WebXR, to understand user actions and their consequences, detailed data on individual movements of the gaze and controller cursors are required, along with the position and rotation of the camera. In addition, information on intentional and unintentional interactions is needed. For example, a click on a transparent ad or a visible ad triggered by a fake invisible cursor is considered unintentional. In contrast, a click on an ad by a fake but visible cursor is regarded intentional. Thus, interactions in identified UI attacks can be intentional or unintentional, highlighting the need for a custom WebXR logger. We implemented our log framework using A-Frame [1], chosen for its simplicity, rapid prototyping capabilities, and active developer community. With built-in WebXR support, it employs a declarative HTML-like syntax and an entity-component-system architecture.

Design & Implementation. Figure 1 illustrates the design of our log framework, featuring a modular structure and reusable components. It consists of four primary log components: ❶ environment scanner, ❷ T_{obj} and DP_{obj} logger, ❸ cursor event logger and ❹ camera and gaze logger.

Single log component can be attached to multiple entities in a scene by its name and schema parameters to add corresponding functionality. These components undergo lifecycle stages such as creation, update, and destruction that can be dynamically managed using lifecycle handlers (e.g., `init()`, `tick()`, `update()`, `remove()`). The `init()` method runs at component initialization, while `tick()` executes at the start of each frame rendering. Thus, a log component's initialization code can be placed in `init()`, and continuous monitoring logic can be implemented within `tick()`.

First, the environment scanner ❶ identifies and assigns the T_{obj} logger and the DP_{obj} logger to the respective entities. Within a dynamic scene, an entity may be removed or detached, leading to the removal of its log component. If the entity reappears, the log component must be reattached, which is handled by monitoring within the `tick()` method.

The T_{obj} logger 2a and DP_{obj} logger 2b track events such as cursor focus initiation (`raycaster-intersected`), focus loss (`raycaster--intersected-cleared`), and clicks, recording the corresponding timestamps, entity IDs, and cursor IDs. Additionally, the DP_{obj} logger monitors position changes based on attack context by listening for `componentchanged` events, logging timestamps, DP_{obj} position, and gaze position. The criteria for considering interactions as intentional or unintentional are also included within this component.

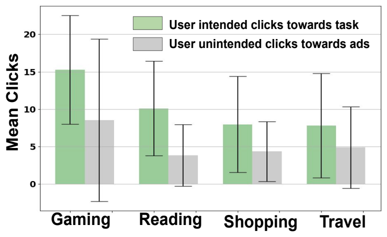
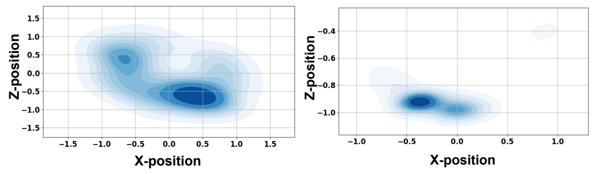
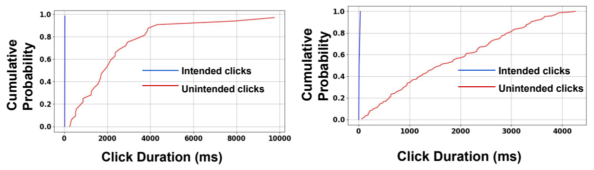


Figure 2: Aggregated mean and standard deviation of clicks.



(a) Gaming (b) Reading

Figure 3: User gaze density, indicating attention.



(a) Attack 🅐 (b) Attack 🅒

Figure 4: CDF of click durations.

For example, for the Visual Overlapping attack, an unclickable T_{obj} is kept in the foreground of DP_{obj}, and any click towards the T_{obj} is forwarded to the clickable DP_{obj}. If the cursor intersects both objects simultaneously, the event is marked as unintended; otherwise, as intended. These events are registered within the `init()` method.

To capture data on simultaneous intersections with multiple entities via a single cursor, the logger incorporates the cursor event logging component ❸. It independently monitors the various events occurring on all cursors (left and right controller cursors and gaze cursor) within the given scene. More specifically, it sets up event listeners in the `init()` function for events like `raycaster-intersection` and `click`. This process includes recording details such as the timestamp, the count of intersected entities, and, for each intersected entity, the intersection point, distance, and its ID.

Lastly, the log framework introduces a camera and gaze logger ❹ to estimate the user's position and attention and determine whether the DP_{obj} is outside or inside the user's focus area. However, logging this information at every frame is computationally expensive and redundant, as it may not always be associated with meaningful interactions. Therefore, it logs this information only during interactions involving any of the cursors, where it captures the timestamp, field of view (FoV), position, rotation, and direction of the camera, along with the gaze position. The log framework assigns these event listeners within the `init()` method. These captured event details help us determine the user interactions within the dark pattern integrated WebXR environment.

Deployment. The logger's modular design and single entry point make it flexible and easy to integrate into any A-Frame app. The developer defines T_{obj} and DP_{obj} based on the context of the app and attaches the entry point of the logger to an entity in the scene. The logger was integrated into all apps, either incorporating an attack or serving as a control group app without attack.

We conducted three experiments per user within a one-hour session, where each user interacted with three selected scenarios and completed an experience survey after each experiment. For each experiment, we recorded the log data and downloaded it into a text file once the user exited the immersive mode in WebXR. During the 100-participant between-subjects user study, we distributed the apps using Glitch [11] platform while remotely serving the app URL on the Meta Quest 2 device using the Meta Quest Developer Hub (MQDH) [8].

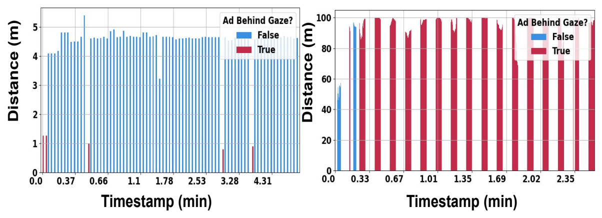
3 Evaluation

User Interactions. Our log framework records user-intended and unintended interactions, along with detailed intersection data such as intersection points, timestamp, and cursor ID, providing valuable insights into user interaction behaviors within the 3D WebXR environment. Using the collected data, we can determine the frequency of intended and unintended interactions, identify the cursors involved, and detect patterns in unintended interactions, such as whether they occur at specific times or are instigated by particular entities. For example, Figure 2 presents the mean and standard deviation of unintended clicks on DP_{obj} and intended clicks on T_{obj} across different apps, using data collected from 100 users. The results reveal notable variation in user behavior within the same app type, supporting the feasibility of capturing varying user interaction patterns. We also observe more intended clicks in the gaming task and fewer in the reading task, highlighting differences in interaction frequency for these apps.

User Attention. Through an analysis of gaze movement data, our log framework also reveals insights into user attention in the 3D space. To illustrate, in Figure 3 we see that user attention is more widely distributed in the gaming app, whereas the reading app shows a single strong fixation point.

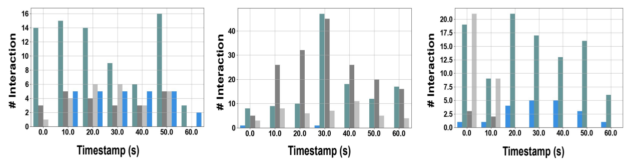
Clicks Uncertainty. The log framework captures data on focus initiation, clicks, and focus removal. By analyzing the time difference between focus removal and clicking, we can infer user uncertainty about their actions. Longer intervals indicate unclear feedback, helping identify problematic UI elements. For example, as shown in Figure 4, intended clicks were shorter in duration for both 🅐 and 🅒, indicating that when users received feedback on their actions, they quickly shifted focus and moved the cursor elsewhere. However, the figure also shows that 🅐 results in prolonged unintended interactions, whereas the logger detects multiple shorter unintended clicks caused by 🅒. Intended clicks are substantially fewer in 🅒, while repeated unintended actions indicate the user's difficulty in interacting with the UI, aligning with the attack objective.

Target Changes. The DP_{obj} logger component in our framework tracks the position changes of DP_{obj} over time along with user gaze positions. Figure 5 compares 🅑 and 🅒 based on positional changes



(a) Attack Ⓒ (b) Attack Ⓑ

Figure 5: Ad object position over time.



(a) Attack Ⓐ (b) Attack Ⓒ (c) Attack Ⓓ

Figure 6: Interactions over time. ⬜ Unintended focus, ▨ Unintended click, 🟩 Intended focus, 🟦 Intended click.

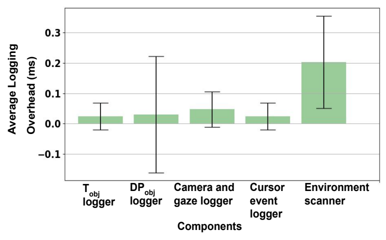


Figure 7: Logging overhead for each component.

of $\texttt{DP}_{\texttt{obj}}$ over time. Specifically, it shows whether the $\texttt{DP}_{\texttt{obj}}$ remains outside the user's area of attention. The figure shows that, unlike Ⓒ, in Ⓑ, the $\texttt{DP}_{\texttt{obj}}$ remains outside the user's attention area for most of the time, tracking them from a large distance behind.

Interaction Trend. Our log framework also facilitates the analysis of user's unintended and intended interaction trends over time. Figure 6 shows a single user's interactions over one minute in an app with a specific attack type. Figure 6(b) shows a vast number of unintended interactions Ⓒ every 10 seconds. Compared to the attack from Ⓐ for the same app (Figure 6(a)), Ⓒ exhibits a substantially lower number of intended clicks throughout the entire time period. As shown in Figure 6(c), our log framework detects a high number of unintended interactions for the attack from Ⓓ during the initial phase of sensitive data extraction.

Performance Analysis. Collecting logs in resource intensive apps can affect performance and immersion. On average, our framework records 62.6 KB logs per minute. The overhead evaluation of our log framework indicates an average overhead of 0.19 ms with a standard deviation of 0.16 ms. According to Figure 7, the environment scanner ❶ has the highest overhead, with significant variability also observed in the $\texttt{DP}_{\texttt{obj}}$ logger ②b. However, the overall overhead incurred by the framework is minimal.

4 Discussion & Conclusions

As detailed in Section 3, the environment scanner ❶ incurs the highest overhead, despite the minimal overall impact of the log framework. This component's `tick()` method, which continuously monitors dynamically generated entities per frame, is resource-intensive due to its comprehensive scene scan. To optimize, the scanner should focus solely on specified, dynamically generated entities, requiring developers to define context-specific rules. In addition, it can also reduce the frequency of event logging to optimize overhead. Furthermore, implementing remote log collection for continuous server-side analysis would be more efficient than post-session data retrieval.

Despite limitations, our log framework offers versatile utility for developers and platforms. Its modular design facilitates easy integration into any A-Frame app. Developers can assess user attention and interaction patterns, enabling strategic entity placement and the detection of unintended interactions with third-party elements, such as ads. Moreover, our evaluation shows the framework's potential for platforms like WebXR [13] to detect malicious activities and warn users of UI manipulation threats in immersive web environments. This enables users to continue or reduce interaction with the affected interface, mitigating loss of autonomy.

For simplicity, the logger was tailored for specific attack types in this study. Future work will extend it for simultaneous detection of multiple attacks and applying machine learning to identify elements like $\texttt{T}_{\texttt{obj}}$ and $\texttt{DP}_{\texttt{obj}}$, reducing developer effort while maintaining user control in web-based immersive platforms.

5 Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This work was partially supported by NSF through grants CNS-2144645 and IIS-2229876. The findings and recommendations in this work are those of the authors and do not necessarily represent the views of the NSF.

References

[1] A-Frame. https://aframe.io/. [Online; accessed 31-Mar-2025].
[2] Google Analytics. https://developers.google.com/analytics. [Online; accessed 31-Mar-2025].
[3] Kaiming Cheng, Arkaprabha Bhattacharya, Michelle Lin, Jaewook Lee, Aroosh Kumar, Jeffery F. Tian, Tadayoshi Kohno, and Franziska Roesner. 2024. When the User Is Inside the User Interface: An Empirical Study of UI Security Properties in Augmented Reality. In *USENIX Security Symposium*.
[4] WebXR Medical Simulation Demo. https://web-xr-med-sim.vercel.app/sim/demo. [Online; accessed: 31-Mar-2025].
[5] Firebase. https://firebase.google.com/. [Online; accessed 31-Mar-2025].
[6] A-Frame Gaming. https://heyvr.io/arcade/games/wackarmadiddle. [Online; accessed: 31-Mar-2025].
[7] Hyunjoo Lee, Jiyeon Lee, Daejun Kim, Suman Jana, Insik Shin, and Sooel Son. 2021. AdCube: WebVR Ad Fraud and Practical Confinement of Third-Party Ads. In *USENIX Security Symposium*.
[8] MQDH. https://developers.meta.com/horizon/documentation/unity/ts-odh/. [Online; accessed: 31-Mar-2025].
[9] Chandrika Mukherjee, Reham Mohamed, Arjun Arunasalam, Habiba Farrukh, and Z Berkay Celik. 2025. Shadowed Realities: An Investigation of UI Attacks in WebXR. In *USENIX Security Symposium*.
[10] US Department of Defense Optimises Immersive Learning Tools. https://www.xrtoday.com/mixed-reality/us-department-of-defence-optimises-immersive-learning-tools. [Online; accessed: 31-Mar-2025].
[11] Glitch Platform. https://glitch.com/. [Online; accessed 31-Mar-2025].
[12] A-Frame Shopping. https://aframe.io/aframe/examples/showcase/shopping/. [Online; accessed: 31-Mar-2025].
[13] WebXR. https://www.w3.org/TR/webxr/. [Online; accessed 31-Mar-2025].