

EXRPOSE: Runtime UI-based Attack Detection in WebXR

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

Aishwarya Devi Akila Pandian
Purdue University
West Lafayette, IN, USA
aakilapa@purdue.edu

Arjun Arunasalam
Florida International University
Miami, FL, USA
aarunasa@fiu.edu

Jiasi Chen
University of Michigan
Ann Arbor, MI, USA
jiasi@umich.edu

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

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

Abstract

WebXR enables users to interact with 3D environments directly through the browsers of their head-mounted displays. These environments use UI properties, such as transparency, and object overlap to create realistic immersive experiences. Recent work has shown that adversaries can exploit these properties to launch UI-based attacks (e.g., input interception via transparent overlays and programmatic click injection). These attacks cause severe harm, such as data theft, malware downloads, and financial loss, without the user's knowledge. Yet, current defenses against UI attacks that rely on post-hoc analysis, trusted configurations, or narrow UI observation suffer from delayed detection, vulnerability to malicious insiders, and incomplete attack coverage.

In this paper, we introduce EXRPOSE, a runtime policy enforcement system for WebXR, which detects UI-based attacks and provides alerts to the user in real time. It operates in three phases. First, it derives security policies from a systematic analysis of UI-based properties and formalizes them in a temporal logic with time constraints to characterize attack states. Second, it inserts code blocks into WebXR applications to capture runtime events and structure the extracted contextual signals into discrete, time-stamped records. Lastly, it evaluates relevant policies against these signal records upon specific trigger events to flag violations.

To evaluate EXRPOSE, we build test environments on a Meta Quest 3 headset with 15 UI attacks across four WebXR applications and diverse scene compositions, such as varying object counts and geometry types. EXRPOSE detects each ground-truth violation with amortized runtime overhead under 1% of the XR frame refresh rate. We additionally conduct a within-subjects user study with 20 participants and show that real-time alerts help users recognize otherwise missed attacks without negative effects on presence or task load. Through this effort, we provide users with contextual runtime observations for secure immersive applications.

CCS Concepts

• **Security and privacy** → **Web application security**; *Usability in security and privacy*.

Keywords

WebXR security, UI-based attacks, runtime monitoring

ACM Reference Format:

Chandrika Mukherjee, Aishwarya Devi Akila Pandian, Arjun Arunasalam, Jiasi Chen, Z. Berkay Celik, and Habiba Farrukh. 2026. EXRPOSE: Runtime UI-based Attack Detection in WebXR. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26)*, November 15–19, 2026, The Hague, Netherlands. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3830454.3846567>

1 Introduction

WebXR enables users to interact with immersive 3D scenes directly through the browser of their head-mounted displays (HMDs), such as Meta Quest 3 [42] and Apple Vision Pro [7]. Since WebXR requires only a browser that supports the WebXR API [63], developers can publish platform-agnostic applications without the need to target specific hardware. Applications range from games [1, 43] and art [4] to fitness [59].

WebXR supports various input modalities, such as controller raycasts, gaze input, and hand tracking, and it exposes a rich set of UI properties (e.g., transparency, object overlap, and synthetic or programmatic input) to construct immersive applications. For example, in an escape room scene, the activation of a lever can programmatically open a door. This mimics real-world cause-and-effect and does not require the user to approach and interact with the door directly.

As the WebXR application ecosystem grows, so does the presence of advertisements (ads). Platforms such as Zesty [66] and heyVR [21] provide SDKs that embed third-party ad content directly within the 3D scene. Recent work [14, 34, 47] has demonstrated several UI-based attacks within this ecosystem, where the ad-service provider, advertiser, or developer can act as the adversary. For example, a malicious advertiser can serve a transparent ad that overlays interactive objects to intercept user clicks and silently redirect the user's browser to a cross-origin webpage [47]. Similarly, a malicious developer can dispatch synthetic (programmatic) clicks on ads outside the user's field of view to inflate ad revenue without any user interaction [14]. These attacks can cause unauthorized navigation, financial fraud through inflated ad clicks, malware downloads, data theft, and reputational damage to benign stakeholders, all while the user remains immersed and unaware.

These attacks exploit the absence of content isolation in WebXR and the spatial properties of 3D scenes. On the standard web, the `<iframe>` element [25] isolates third-party content from the host



This work is licensed under a Creative Commons Attribution 4.0 International License. *CCS '26, The Hague, Netherlands*

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2871-6/2026/11
<https://doi.org/10.1145/3830454.3846567>

page. WebXR has no equivalent mechanism because the entire 3D scene renders within a single HTML `<canvas>` element. Therefore, ad SDKs inject ads as standard scene entities that share the same scene graph, render context, and JavaScript execution environment as first-party objects. However, a defense that provides only execution isolation is not sufficient. This is because attacks also exploit the spatial properties of 3D scenes, where objects at different depths can occlude, overlap, or intercept input intended for other objects.

On the 2D web, UI-based attacks are well studied [6, 24, 53], and defenses such as `iframe` sandbox mechanisms [25], Content Security Policy [16], and X-Frame-Options [52] are built around the web's document and frame model. These defenses protect frame boundaries, but WebXR attacks arise from how 3D objects appear, overlap, hide, and receive input inside the immersive scene. Thus, they do not directly address attacks caused by transparency, depth, occlusion, field of view, or synthetic input. Runtime detection techniques from adjacent domains (2D web, mobile, IoT) [9, 13, 51] also fall short because they do not model the spatial and interaction semantics of WebXR.

Within immersive environments, existing defenses [34, 36, 55] partially address the problem. That is, they rely on trustworthy developer configurations, support only post-hoc forensic analysis, or target a limited set of exploitable properties. Static analysis techniques are also insufficient, because they cannot account for dynamic changes in object properties and user interactions that occur only at runtime. In addition, a static evaluation of all possible states can either produce an excessive number of false positives or fail to detect violations that depend on specific runtime conditions. Users therefore remain unprotected during the immersive experience, with no mechanism to detect or warn them as attacks occur. This motivates a formally grounded runtime system that detects the deceptive manipulation and malicious use of UI properties.

In this paper, we introduce `EXRPOSE`, a runtime detection system for WebXR that identifies UI-based attacks and alerts users to unintended interactions caused by the manipulation of security-sensitive UI properties. We first systematically evaluate how deceptive manipulations of security-sensitive UI properties cause unintended interactions. For each interaction pattern, we identify the events and contextual predicates that determine whether user intent has been subverted. We combine these specific events and predicates to formulate security policies. We formalize each policy with signal temporal logic (STL) [37] to specify expected safe behavior. For example, a policy requires that a clicked object is not transparent and does not occlude a visible target, or that no unauthorized cross-origin redirect occurs. Second, we instrument the application with code blocks to capture this information at runtime. Lastly, during execution, we evaluate the captured events and their corresponding timestamped predicates against the relevant security policies. If an application state violates a policy, `EXRPOSE` records the timestamped violation event and alerts the user. To the best of our knowledge, this is the first work to formalize runtime policy enforcement for UI-based attacks in WebXR.

To evaluate `EXRPOSE`, we construct custom browser-based immersive environments on a Meta Quest 3 headset. We test 15 UI attacks across four WebXR applications representing diverse use cases. We evaluate the performance overhead of `EXRPOSE` under

varied scene compositions (e.g., the number of 3D objects, the number of intersected entities, and the geometry type) and in production WebXR applications. `EXRPOSE` correctly identifies all ground-truth policy violations across the $15 \times 4 = 60$ scenarios. The total per-event overhead averages under 5 ms. Even in scenes with the highest object counts and cursor intersections, this overhead remains within the 13.89 ms frame interval at 72 Hz, so `EXRPOSE` drops no frames. Because events occur at discrete interaction points, the amortized overhead stays below 1% (at one event per second).

To evaluate the usability of `EXRPOSE` and its impact on the immersive experience, we conduct a within-subjects user study with 20 participants. Each participant interacts with a real-world game scene that integrates four UI-based attacks. We test two conditions: with and without `EXRPOSE`. We collect post-experiment surveys and conduct 12-15 minute interviews to assess threat comprehension after the participants view the warnings. `EXRPOSE` causes no statistically significant impact on user presence or task load. 70% of participants comprehended the purpose of the warnings. After learning about the violations in the without-`EXRPOSE` condition, all participants preferred receiving alerts from `EXRPOSE`, with several indicating a desire to avoid risks such as financial or data theft.

In summary, we make the following contributions:

- We introduce `EXRPOSE`, a runtime detection system for WebXR that identifies UI-based attacks during the immersive experience. `EXRPOSE` formulates security policies from a systematic evaluation of security-sensitive UI properties. It formalizes these policies with signal temporal logic (STL), instruments the application to capture runtime events, and evaluates the application state to detect violations.
- We evaluate `EXRPOSE` on 15 distinct UI-based attack scenarios across four different WebXR applications, and the system correctly identifies all ground-truth policy violations. Across diverse scene compositions, the average per-event overhead remains under 5 ms and within a single frame interval (13.89 ms at 72 Hz), even at the maximum tested object counts and cursor intersections.
- We conduct a within-subjects user study with 20 participants. In a real-world application that integrates four UI-based attacks, `EXRPOSE` detects violations with no statistically significant impact on user presence or task load. Furthermore, participants prefer to receive alerts from `EXRPOSE` rather than experience attacks without awareness.

2 Background

WebXR. A W3C standard, WebXR enables immersive VR and AR experiences directly in the web browsers of head-mounted displays (HMDs) such as Meta Quest 3 [42], HoloLens 2 [22], and Apple Vision Pro [7], without requiring a native application. Browser support varies, as Chromium-based browsers (e.g., the Meta Quest Browser, Microsoft Edge) ship WebXR enabled by default, whereas in Firefox it remains an experimental feature that must be explicitly enabled via a browser preference, and Safari on Apple Vision Pro enables it by default only since visionOS 2. Through the WebXR Device API, web applications access headset displays, controllers, and spatial tracking. WebXR applications are typically written in JavaScript and rendered with WebGL.

Users navigate to a WebXR application by entering its URL, then transition into a 3D immersive scene by clicking an “Enter VR/AR” button, where they experience six degrees of freedom (6DoF) and interact using controller raycasters, gaze cursor, or hand-tracking. Developers build these experiences using frameworks such as A-Frame [2], Three.js [58], and Babylon.js [8], with minimal effort to support both AR and VR modes. Among the available frameworks, A-Frame, built on Three.js, is widely adopted and provides a declarative, HTML-based entity-component architecture with custom elements (e.g., `<a-scene>`, `<a-entity>`) and reusable components for behavior, appearance, and interaction. A key architectural difference from the standard web is that WebXR renders 3D scenes within an HTML `<canvas>` element, which displays only a single scene at a time and provides no `<iframe>`-like isolation boundary between different-origin content. As a result, any third-party content (e.g., ads) shares the main scene’s rendering context and can freely modify the shared environment.

UI Properties. The immersiveness of WebXR applications depends on a set of UI properties that control how objects appear, respond to input, and interact with the user. Unlike the 2D web, WebXR applications place virtual objects in 3D space around the user instead of keeping them fixed on a flat screen. This allows interfaces to appear at different distances, directions, and orientations, making them feel like part of the immersive scene. For example, transparency in WebXR allows objects such as glass, water, fog, or translucent UI panels to blend with the surrounding 3D scene, making the environment feel more natural, layered, and spatially immersive. Objects can be placed at different depths, aligned with other objects, arranged around the user, or positioned outside the user’s current field of view, enabling richer spatial scene construction.

Programmatic behavior allows a user’s interaction with one object to update or animate other objects. This supports responsive cause-and-effect interactions that make the environment feel dynamic and interactive. Scene entry and exit events allow applications to change their UI between 2D browser mode and immersive 3D mode. These events can alter the visibility, layout, or availability of controls. WebXR also supports multiple input modalities. Gaze input enables hands-free interaction, and controller input provides precise targets. This differs from the mouse and keyboard input of the standard web. Together, these properties increase the realism, responsiveness, and spatial presence of the WebXR environment beyond the capabilities of conventional 2D web interfaces.

STL. Signal Temporal Logic (STL) [37] is a specification language for expressing temporal properties over real-valued signals. Unlike classical temporal logics that operate over boolean traces, STL supports predicates of the form $\mu(s(t)) \geq c$, where $s(t)$ is a signal value at time t , μ is a real-valued function over signals, and c is a threshold constant. STL formulas are constructed using standard temporal operators: $\Box$ (always/globally), $\Diamond$ (eventually), $\mathcal{U}$ (until), and boolean connectives. A key feature of STL is its *quantitative semantics*, in which the *robustness* value quantifies how strongly a signal trace satisfies or violates a given formula. A positive robustness indicates satisfaction; a negative robustness indicates a violation. The magnitude reflects the margin of satisfaction or violation, providing a continuous measure rather than a binary verdict. PSY-TaLiRo [57] is a Python toolbox for testing and verification

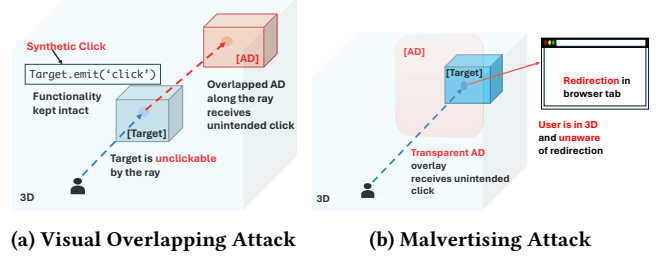


Figure 1: Examples of UI-based attacks in WebXR [47].

of cyber-physical systems that supports STL robustness computation. It provides a unified interface for specifying STL requirements and supports multiple robustness computation backends, including RTAMT [65], which implements dense-time STL monitoring.

Definitions. We use the following terms throughout the paper. A *security-sensitive UI property* (e.g., transparency, programmatic input) serves a legitimate purpose in an immersive scene but can also be exploited maliciously. An *unintended interaction* occurs when a user’s intended action is blocked or intercepted, or when an additional action is triggered without the user’s knowledge or consent. A UI attack induces one or more unintended interactions while concealing its presence. A *policy* is a formal specification of expected safe behavior with respect to a security-sensitive UI property and is violated whenever an unintended interaction occurs at runtime. An *event* is either a user interaction (e.g., click, focus) or a runtime state change (e.g., object creation, position change, opacity mutation, redirection). A cursor is *valid* if it exists and is generated by code from the same origin as the main scene. A *predicate* is a boolean, real-valued, or categorical variable derived from the scene state at an event (e.g., an object’s opacity, whether an object is the first intersected by a cursor, or an object’s raycasting class). A *first-party* object is created by the application developer; a *third-party* object is injected by external code (e.g., an ad script).

3 Motivation and Threat Model

Motivation. The same UI properties that make WebXR immersive (e.g., transparency, overlapping objects, and synthetic input) also create its attack surface. Prior work [14, 34, 47] has demonstrated UI attacks that exploit these security-sensitive properties to implement deceptive patterns [39, 40] in the advertising ecosystem, where any stakeholder in the ad chain may act as an adversary. For example, in the visual overlapping attack (Figure 1a) [47], a malicious developer places an ad behind an unclickable target so the two overlap from the user’s view. A click on the seemingly interactive target passes through to the ad, while a synthetic click on the target preserves application functionality, generating illegitimate ad revenue at the advertiser’s expense. Other attacks exploit regions outside the user’s awareness (e.g., a browser redirect while the user remains immersed in the 3D scene, Figure 1b), block or intercept the user’s intended actions, or exfiltrate private information.

User studies show these attacks are highly effective because the user’s attention is confined to the immersive view; users typically fail to notice the manipulation, redirections on the auxiliary browser screen, or interactions outside their field of view [47]. These attacks

Table 1: Existing defenses in immersive environments.

Defense	Mechanism	Overhead	Coverage and Limitations
AdCube [34]	Ad sandboxing and visual confinement; trusts the developer/publisher	236 ms load; 1.6% FPS drop	Misses developer-driven UI attacks (e.g., Input Forgery [14], Visual Overlapping [47])
RealityCheck [55]	Provenance-based post-hoc investigation	1.3–2.9%; query ~2–3 ms	Post-hoc root-cause analysis; no runtime UI-attack detection
AutoVR [29]	Offline automated UI exploration	222.40 s/app	Finds sensitive data-flow exposures; no runtime enforcement
Liu et al. [36]	Periodic runtime scene scanning	FPS drops sharply with scan rate/object count	Detects only same-space and invisibility properties

harm every stakeholder. Users suffer sensitive data theft, malware downloads, and financial loss; advertisers pay for inflated clicks and impressions; publishers lose their audience when intended actions are intercepted; and ad-service providers lose publisher clients whose applications are degraded.

Analogous UI attacks have been studied on the 2D web [6, 24, 53], where defenses such as iframe sandboxing [25], Content Security Policy (CSP) [16], and X-Frame-Options [52] isolate or restrict embedded content through document and frame boundaries. These assumptions do not hold in WebXR, where interactive objects are placed, rendered, and triggered inside a shared 3D scene; the problem is not framed content but the spatial and interactive behavior of objects in the immersive scene. A limited number of defenses have been proposed for immersive environments (Table 1); however, they trust developers who can themselves act as adversaries [14, 47], investigate attacks only post-hoc after an external alert [55], operate offline [29], or monitor only two of the 14 known security-sensitive properties [36]. To enable safe adoption of WebXR, a runtime detection system grounded in formal specifications is needed that covers the full space of known UI attacks and alerts users to violations.

Threat Model. Consistent with prior work [14, 34, 47], we consider an adversary that can manipulate JavaScript-accessible properties of objects in a WebXR scene. This can be: (1) A malicious *developer* who controls the application code and scene layout, and may position objects to occlude or overlay other content, manipulate transparency to intercept user interactions, or dispatch synthetic input events. (2) A malicious *third party* with edit access to the scene may modify object properties, inject new objects, register event listeners, trigger URL redirections, or dispatch synthetic input events through an included library, SDK, or embedded script.

We illustrate both adversaries using the advertising ecosystem as an example. A malicious developer (publisher) may dispatch synthetic clicks on ads outside the user’s field of view to inflate engagement metrics [14]. A malicious ad-service provider (third-party) can inject a fake controller that captures the same events as the real controller and uses the resulting clicks to trigger additional clicks on ads [34]. A malicious advertiser may serve partially or fully transparent ads that interfere with user interactions, causing unintended clicks that trigger cross-origin redirects in the auxiliary

browser screen [47] to boost its website ranking or gain first-party cookie access.

The adversary’s goal is to cause harm to users, including data theft, malware downloads, and financial loss. These UI manipulations also harm other stakeholders through fraudulent engagement and degraded user experience. For example, click fraud by a developer or ad service provider deprives a legitimate advertiser of genuine engagement and inflates their costs. We focus on UI manipulation attacks within the WebXR scene through JavaScript-accessible properties, excluding browser or OS vulnerabilities, physical-world manipulation, and side-channel inference. We assume EXRPOSE is integrated by a trusted platform provider such as a WebXR framework, browser, or operating system.

4 Design Challenges

Realizing runtime detection of UI-based attacks in WebXR applications, with alerts as violations occur, raises four design challenges.

C1: Lack of Formal UI Specification. WebXR lacks a formal specification defining how security-sensitive UI properties should be used in immersive applications. Without such a specification, differentiating between benign and malicious events is difficult. For example, flagging all transparent elements would also flag legitimate uses such as glass or water effects, producing spurious violations. No existing ad library or platform provides guidelines for the correct use of these properties to avoid unintended interactions, where a user’s action is blocked, intercepted, or triggered without their knowledge. An existing taxonomy of UI-based attacks in WebXR [47] categorizes the attacks but does not formally define when a security-sensitive property causes an unintended interaction.

C2: 3D Spatial Complexity. Unlike the 2D web, where an iframe provides a flat, bounded region whose contents cannot extend beyond its boundaries, a WebXR scene embeds every object in a shared 3D space. Virtual objects at different depths can occlude one another, overlap along a cursor’s ray, or sit outside the user’s field of view while still receiving events. Determining whether a click or focus event landed on the intended object requires reasoning about the full 3D spatial context, including which object is nearest along the cursor’s ray, what visible objects lie behind or in front of it, whether the object is inside the user’s field of view, and whether it shares its shape and position with another object. For instance, detecting the visual overlapping attack (Figure 1a) requires identifying that the nearest object along the cursor is an unclickable target, that the ad behind it receives the click, and that the target receives a synthetic click to preserve functionality. These spatial complexities require detection mechanisms that go beyond the 2D containment checks sufficient on the standard web.

C3: Real-Time Detection. The system must detect violations as they occur during the immersive experience, since delayed detection leaves users exposed to the attack’s consequences. XR headsets operate at refresh rates of 72–120 Hz, yielding per-frame budgets of 8–14 ms, and any detection overhead must remain within this budget to avoid dropped frames or perceptible lag. This requires an event-driven architecture that performs work only when relevant interactions occur.

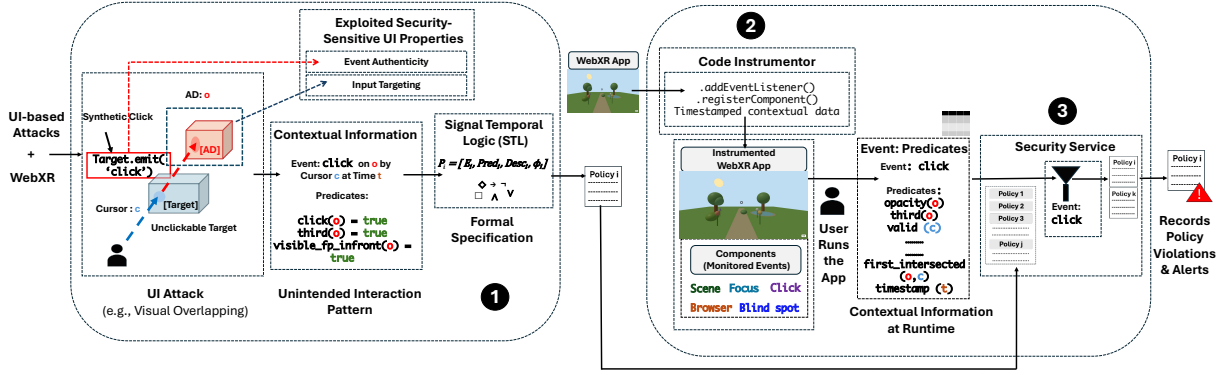


Figure 2: Overview of EXRPOSE's architecture.

C4: Preserving Application Behavior. The detection system must observe the application without altering its behavior. Monitoring events, scene state, and user interactions must not modify event propagation, alter the rendered scene, or disrupt the user experience, as any such interference would compromise the validity of detected signals.

5 Approach Overview

Figure 2 presents an overview of EXRPOSE, our runtime detection system for UI attacks, which operates in three stages: deriving formally defined security policies (1), instrumenting the application to capture runtime signals (2), and evaluating those signals against the policies via a security service (3).

To derive the policies (1), we systematically analyze 15 UI attacks [14, 34, 47], identifying how each exploits security-sensitive UI properties to induce unintended interactions. For each interaction, we extract the events at which it can be observed (e.g., click, redirection) and the predicates that characterize the violation condition (e.g., opacity below a threshold, cross-origin URL) (addressing C1). The predicates capture 3D spatial relationships, such as whether the target is the first-intersected object along the cursor's ray and whether visible objects lie behind or in front of it (addressing C2). We formalize each unintended interaction as a policy in Signal Temporal Logic (STL), yielding 15 policies that cover all 15 attacks.

To capture the runtime information required by the policies, our code instrumentor (2) adds code blocks to the application before execution, registering listeners that extract timestamped predicate values only when a relevant event occurs (addressing C3). The instrumented code does not modify event propagation or alter the rendered scene (addressing C4). Finally, the security service (3) receives the signal records during execution and evaluates each record against only the policies associated with its triggering event (e.g., a click triggers seven policies, a redirection two). For each, the service computes the STL robustness, where a negative value indicates a violation.

6 EXRPOSE System

We present the design of EXRPOSE, a runtime detection system for UI-based attacks in WebXR.

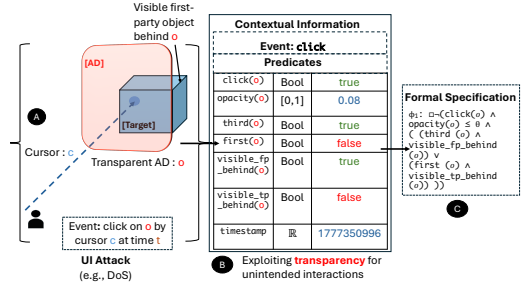


Figure 3: Security policy construction.

6.1 Security Policies

We derive security policies from UI attacks in WebXR, i.e., attacks that manipulate what users perceive or how their input is interpreted at the API level. A systematic review of the literature at top-tier security venues yielded three papers [14, 34, 47] covering 14 attacks. We exclude attacks involving emotional manipulation, physical-world safety risks, multi-user settings, and multi-sensor exploitation, as they do not target WebXR UI behavior alone.

Figure 3 illustrates policy construction using the Denial-of-Service (DoS) attack [14] as an example. We analyze how attacks exploit security-sensitive UI properties to induce unintended interactions (A), identify the events and predicates needed to characterize each unintended interaction (B), and formalize each policy in Signal Temporal Logic (STL) (C).

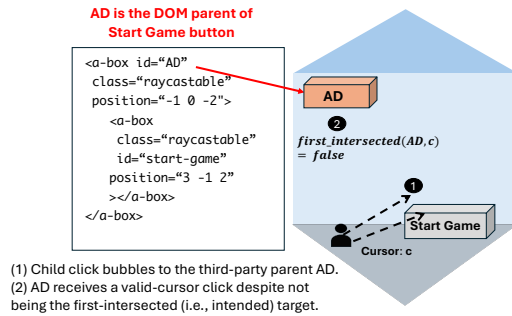
From Attacks to Unintended Interactions. Prior work [14, 34, 47] has identified 14 UI attacks in WebXR that manipulate security-sensitive properties such as transparency, object overlap, and synthetic input to cause unintended interactions, i.e., a user's intended action is blocked or intercepted, or an action is triggered without the user's knowledge or consent. For example, in the DoS attack [14] (Figure 3 A), a transparent clickable object placed over the user's intended target receives the click instead, and the user notices neither the transparent layer nor the interception. While DoS exploits this single interaction, Malvertising [47] and Intercepting User Inputs [14] exploit the same transparent-overlay interaction but additionally trigger cross-origin redirections or programmatically submit extracted user data, actions the user neither initiates nor

Table 2: Summary of the 15 security policies with their triggering events, STL formulas, descriptions, and the attacks each policy detects.

P_i	Event Set E_i	STL Formula φ_i	Description $Desc_i$	UI-based Attacks
P_1	{click}	$\Box \neg (click(o) \wedge opacity(o) \leq \theta \wedge ((third(o) \wedge visible_fp_behind(o)) \vee (first(o) \wedge visible_tp_behind(o))))$	A click on a low-opacity object must not occur when a cross-party visible object exists behind it (e.g., a first-party target occluded by a third-party ad).	Denial-of-Service [14], Intercepting User Inputs [14], Malvertising [47]
P_2	{click}	$\Box \neg (click(o) \wedge (\neg valid(c) \vee \neg first_intersected(o, c)))$	A click must not be accepted if the cursor is invalid (absent cursor or generated by cross-origin code) or the target is not first-intersected by the cursor.	Visual Overlapping [47], Sequential Rendering [47], Input Forgery [14], Intercepting User Inputs [14], Event Bubbling*
P_3	{click}	$\Box \neg (click(o) \wedge first_intersected(o, c) \wedge out_fov(o) \wedge third(o))$	A click must not land on a first-intersected third-party object outside the user's field of view.	Controller Cursor-Jacking [34]
P_4	{redirect, replace}	$\Box \neg ((replace_window(id) \vee window_open(id)) \wedge \neg same_origin(id))$	A cross-origin redirection must not occur.	Malvertising [47], GUI Switch [47]
P_5	{click}	$\Box \neg (click(o) \wedge third(o) \wedge visible_fp_infront(o))$	A click must not land on a third-party object when a visible first-party object is in front of it.	Visual Overlapping [47]
P_6	{click, focus}	$\Box \neg (cursor_opacity(c) \leq \theta \wedge cursor_action(c))$	A cursor with opacity at or below the threshold must not click or focus on any object.	Gaze Cursor-Jacking [34], DoS through Overriding [47]
P_7	{redirect, replace}	$\Box ((scene_entry \vee scene_exit) \rightarrow \neg \Diamond_{[0, T]} ((replace_window(id) \vee window_open(id)) \wedge \neg same_origin(id)))$	A cross-origin redirect must not occur within T ms of a scene entry or exit.	Abuse of Auxiliary Display [34], Malvertising [47] [†] , GUI Switch [47]
P_8	{obj_create, pos_change}	$\Box \neg (created_or_moved(o) \wedge third(o) \wedge media(o) \wedge opacity(o) > \theta_f \wedge first(front(o)) \wedge \neg clickable(front(o)) \wedge opacity(front(o)) < \theta_f \wedge ((\neg related(o, front(o)) \wedge loaded_after(o, front(o)) \vee anc_of(o, front(o))))$	A non-transparent third-party media element (i.e., image, video, or text) must not be obscured by a transparent, non-clickable first-party object that either is unrelated and loaded earlier, or is a DOM descendant of the media element.	Sequential Rendering [47]
P_9	{click}	$\Box \neg (click(o) \wedge third(o) \wedge same_shape_space(o))$	A click must not land on a third-party object that shares the same shape, size, and position as another object.	Clickjacking [14]
P_{10}	{iframe_create}	$\Box \neg (in_xr \wedge iframe_created(id))$	An iframe must not be created during an immersive XR session.	Abuse of Auxiliary Display [34]
P_{11}	{screenshot}	$\Box (screenshot \rightarrow last_click_delta < T)$	A programmatic screenshot must not be captured unless a user click occurred within the preceding T ms.	GUI Switch [47]
P_{12}	{focus}	$\Box \neg (focus(o) \wedge opacity(o) \geq \theta \wedge first_intersected(o, c) \wedge class(o) \neq class(c))$	A cursor must not focus on a visible first-intersected object if the object and cursor belong to different raycasting classes.	DoS through Overriding [47]
P_{13}	{click}	$\Box \neg (click(o) \wedge will_bubble(o) \wedge third(parent(o)))$	A click must not propagate via DOM event bubbling to a third-party parent.	Event Bubbling*
P_{14}	{material_change}	$\Box \neg (mutated(o) \wedge opacity_changed(o) \wedge opacity(o) \leq \theta \wedge exists(o) \wedge third(o) \wedge \neg out_fov(o))$	A third-party object that remains in the scene and within the user's field of view must not have its opacity at or below the threshold after a runtime mutation.	Object Erasure [14]
P_{15}	{blind_spot}	$\Box ((third(o) \wedge out_fov(o)) \rightarrow out_fov_duration(o) < T)$	A third-party object must not remain outside the user's field of view beyond T ms.	Blind Spot Tracking [34], Input Forgery [14]

Note: Predicate names abbreviate first-party as *fp* and third-party as *tp* (e.g., *visible_fp_behind*).[†] P_7 fires only within T ms of a scene entry or exit.

* Newly identified attack.

**Figure 4: Event Bubbling attack.**

perceives. Our analysis further identifies an additional unintended interaction caused by event bubbling in the DOM hierarchy, which we exploit in a new *Event Bubbling Attack*.

Particularly, when one object is a child of another, a click on the child propagates to the parent, which receives the same click event carrying the context of the original interaction, including the initiating cursor; this distinguishes it from programmatic clicks, which lack an initiating cursor. In this attack, a malicious developer makes an ad the DOM parent of a legitimate interactive object (Figure 4), so every click on the object propagates to the ad. Because parent and child objects can occupy arbitrary world-space positions in 3D, the ad appears visually separated, or even outside the user's field of view, yet receives a click whenever the user interacts with the child. The attack thus induces two unintended interactions, as the user's click propagates to a third-party parent via the DOM

hierarchy, and the propagated click, though carrying a valid cursor, lands on an ad that is not the first object intersected along the cursor’s ray.

As these examples show, the relationship between attacks and unintended interactions is *many-to-many*: multiple attacks may share the same unintended interaction pattern, and a single attack may induce multiple unintended interactions. Analyzing all 15 attacks yields 15 distinct unintended interaction patterns. We refine these patterns with contextual conditions to reduce spurious violations. For example, placing a transparent object in front of a visible one is a common legitimate design choice (e.g., for water or glass effects); if both objects are first-party, flagging it would be spurious. We therefore restrict policies to objects belonging to different parties (e.g., one first-party and one third-party), as shown in Figure 3 ③, consistent with the cross-party manipulation in the original attack definitions. To formalize each pattern as an enforceable policy, we next identify the events and predicates that characterize it.

Events and Predicates. We organize the contextual information that characterizes each unintended interaction pattern into events and predicates: an event determines *when* a policy is evaluated, and a predicate is a signal sampled at that moment, determining *what* the policy tests.

Events are runtime triggers at which a policy may be violated. Each policy subscribes to the subset of events that triggers its evaluation. We identify 12 event types, forming the event universe $\mathcal{E}$: *click* (user click or select), *focus* (cursor dwells on or fuses with an object), *redirect* (window.open call), *replace* (URL replacement, e.g., via location.replace), *scene_entry* and *scene_exit* (immersive session starts or ends), *obj_create* (object added to the scene), *pos_change* (object repositioned), *material_change* (material properties mutated, e.g., opacity), *iframe_create* (iframe inserted into the DOM), *screenshot* (programmatic canvas export), and *blind_spot* (third-party object leaves the field of view). These types cover all events exploited in UI-based attacks.

Predicates are boolean, real-valued, or categorical signals derived from the scene state at the time of an event. Throughout, o denotes the triggering event’s target object, c the cursor generating it, and id a navigation identifier; *front*(o) and *parent*(o) return the object in front of o and its DOM parent, allowing predicates to compose (e.g., *third*(*parent*(o))). Each event occurrence is also recorded as a boolean predicate (e.g., *click*(o)), so a policy can reference both its trigger and earlier events. *scene_entry* and *scene_exit* trigger no policy directly; they are captured so policies can reason about session timing. *mutated*(o) marks that a *material_change* event fired on o , and property-specific predicates (e.g., *opacity_changed*(o)) identify what changed. The remaining state predicates fall into five categories: *visibility*, such as *opacity*(o) (material opacity, continuous on $[0, 1]$); *spatial*, such as *first_intersected*(o, c) (the object is nearest along the cursor’s ray) and *visible_fp_behind*(o) (a visible first-party object exists behind the target); *structural*, such as *will_bubble*(o) (a click propagates to a parent via DOM event bubbling); *identity and attributes*, such as *third*(o) (party membership), *valid*(c) (the cursor exists and is generated by same-origin code), and *media*(o) (the object is a media element, i.e., image, video, or text); and *temporal*, such as *last_click_delta* (time since the most

recent user-generated click) and *out_fov_duration*(o) (time a third-party object has remained outside the field of view). Table 2 lists the predicates each policy evaluates and describes their meaning.

For example, to characterize the DoS attack’s unintended interaction pattern (Figure 3 ⑤), the triggering event is *click* and the relevant predicates are *visible_fp_behind*(o), checking for a visible first-party target behind the overlay, *opacity*(o), capturing the overlay’s near-zero opacity, and *third*(o), marking it as a third-party ad. Together, these predicates indicate that the user’s click landed on an invisible third-party overlay rather than the visible first-party target the user intended.

Policy Formalization. We formalize each unintended interaction as a policy that the security service can evaluate at runtime (Figure 3 ⑥). We use STL [37] because it operates over real-valued signals, matching the continuous nature of predicates such as opacity and time intervals. Unlike LTL [50] and MTL [31], which produce binary verdicts, STL provides quantitative robustness, a measure of how far a signal lies from the violation boundary, enabling graded assessment of violation severity. Each policy P_i is a four-tuple:

$$P_i = [E_i, Pred_i, Desc_i, \varphi_i] \quad (1)$$

where $E_i \subseteq \mathcal{E}$ is the set of triggering events, $Pred_i$ is the set of relevant predicates, $Desc_i$ is a natural-language description of the violation condition, and φ_i is the STL formula specifying the safety property; a negative robustness of φ_i indicates a violation. This yields the 15 security policies in Table 2.

These policies follow three STL patterns. Twelve policies (P_1 – P_6 , P_8 – P_{10} , P_{12} – P_{14}) use an *invariant negation* pattern, $\Box \neg(\cdot)$, stating that a conjunction of predicates must never hold (e.g., a click on a transparent third-party object with a visible first-party object behind it, in P_1). P_7 uses a *bounded response* pattern, $\Box(A \rightarrow \neg \Diamond_{[0,T]} B)$, forbidding a cross-origin redirect within T ms of a scene transition. P_{11} and P_{15} use a *prerequisite* pattern, $\Box(A \rightarrow B)$, requiring a condition to hold whenever a triggering action occurs (e.g., a screenshot requires a user click within the preceding T ms, in P_{11}).

6.2 Code Instrumentor

The security policies specify the events and predicates required to detect each violation. These signals must be captured at runtime with minimal latency and without altering application behavior. We therefore develop a code instrumentor that adds code blocks to the application’s source before execution; these blocks register listeners that extract predicate values at each relevant event.

Design Principles. The instrumentor follows two design principles. First, *event-driven monitoring*: signals are captured on application-level events rather than by continuous per-frame scanning, so overhead scales with the frequency of security-relevant events. For P_{12} and P_{15} , which detect conditions that generate no application-level event, the instrumented code performs rate-limited periodic checks but extracts predicate values only on a relevant state change. Second, *observe-only instrumentation*: the instrumented code reads scene state and event metadata; it does not modify event propagation, object properties, or the rendered scene. Scene entities are discovered through scene-graph attachment events and a one-time sweep at initialization, without manual annotation. The instrumentor comprises five components, one per signal category: click,

Table 3: Signal categories, the event types they capture, the policies each event triggers, and the capture mechanism.

Category	Event Type	Policies	Capture Mechanism
Click	click	$P_1-P_3, P_5, P_6, P_9, P_{13}$	Listens for click events; casts an observation ray to obtain the order of intersected objects
Focus	focus	P_6, P_{12}	Listens for focus events; for P_{12} , periodically casts a ray that ignores raycasting-class filtering
Browser	redirect, replace	P_4, P_7	Intercepts window.open calls and URL replacements
	iframe_create	P_{10}	Observes DOM mutations for inserted iframes
	screenshot	P_{11}	Intercepts programmatic canvas exports
Scene	obj_create, pos_change	P_8	Listens for scene-graph attachments; casts a ray from the camera to the new object
	material_change	P_{14}	Monitors material property mutations
	scene_entry, scene_exit	—	Records session transitions for the trace
Blind spot	blind_spot	P_{15}	Periodically samples third-party objects' field-of-view status

focus, browser-level (redirect, replace, iframe_create, screenshot), scene-level (obj_create, pos_change, material_change, and session transitions), and blind-spot monitoring (Table 3).

Signal Extraction. Table 3 maps each event type to the policies it triggers and the mechanism that captures its signals. Most predicates are read directly from the event and the scene state when a listener fires; the rest are computed within the listener, as described below for the main cases. On a click, *first_intersected(o, c)* checks whether the clicked object is the first object the triggering cursor intersects, exposing synthetic clicks that are not aimed at the target. Spatial predicates on clicks (e.g., *visible_fp_behind(o)*, *visible_fp_infront(o)*) additionally require the order of all objects along the cursor's ray. The instrumented code therefore casts a separate observation ray from the cursor's origin and walks the intersection list forward and backward from the target. This ray triggers no focus or intersection events, so observation does not alter application behavior. The same list serves *same_shape_space(o)*, which compares the target's bounding box against the other intersected objects, and *out_fov(o)* is computed from the angle between the camera's forward direction and the camera-to-target vector.

For P_{12} , the framework generates no focus event when the cursor and object belong to different raycasting classes. The instrumented code therefore periodically casts its own ray and records whether *class(c)* matches *class(o)* for the first intersected object. For P_{15} , a third-party object leaving the field of view likewise generates no application-level event. At each sampling interval, the instrumented code checks whether each third-party object lies outside the field of view and accumulates *out_fov_duration(o)*, the time the object has remained there. For P_{14} , mutations to objects outside the field of view are not flagged, as the user cannot perceive them. The remaining predicates are computed as specified by their descriptions in Table 2.

6.3 Security Service

The instrumented application captures signal records at each relevant event, but these records alone do not indicate whether a violation has occurred. The security service evaluates each record

against the formalized policies to make this determination. EXRPOSE is designed as a locally deployed system, with the security service running on the same device as the instrumented application, under the control of the platform provider (e.g., the WebXR framework, browser, or operating system).

Selective Policy Checking. Each signal record carries a timestamp, the triggering event type, and the predicate values captured at that event. Rather than evaluating all 15 policies at every event, the service evaluates only the policies whose event set E_i contains the record's event type (Table 3). Click events trigger the largest subset with seven policies, focus, redirect, and replace events trigger two each, and every other event type triggers one. This bounds evaluation at seven policies per event, reducing runtime overhead.

Policy Evaluation. For each selected policy P_i , the service extracts the predicate values $Pred_i$ from the signal record and computes the STL robustness of ϕ_i . A negative robustness indicates a violation. The robustness magnitude indicates how far the signal values lie from the violation boundary, which can be used to prioritize violations or calibrate policy thresholds. The service then notifies the instrumented application. In deployment, the service can be configured to (1) alert the user, (2) block the violating action and alert the user, or (3) alert the user and let them decide how to proceed.

7 Implementation

We implement EXRPOSE using A-Frame [2] and Three.js [58]. The policies and predicates rely on concepts common to any 3D scene framework (opacity, first-intersected object, party membership, field of view), making the core logic portable.

Code Instrumentor. The instrumentor, approximately 1,755 lines of JavaScript, loads as an ES module [17] and attaches to `<a-scene>`, installing listeners and patches before application or third-party code executes. It enumerates static entities with `querySelectorAll` at scene load and listens for A-Frame's `child-attached` event for later additions. It also patches browser and framework APIs, wrapping `window.open`, logging anchor navigations with a capture-phase document listener, monitoring inserted `<iframe>` nodes with a `MutationObserver`, and wrapping `canvas-export` APIs (e.g., to `DataURL`, `toBlob`) to detect programmatic screenshots. Party membership is assigned at DOM insertion time by patching insertion APIs and attributing each element to its originating script via `Error().stack` inspection. Publisher-created elements are tagged first-party, and all others third-party.

Algorithm 1 shows click signal extraction, which serves seven policies. It first checks whether the triggering cursor exists and is generated by same-origin code (line 1) and whether the target is the first object the cursor intersects (line 2), detecting synthetic clicks. It then casts a dedicated ray from the cursor's origin (lines 3–5), identifies the first visible object before and after the target along the ray, from which the cross-party visibility predicates derive (lines 6–7), and computes the remaining signals (lines 8–10). The ray does not trigger any events, leaving application behavior unchanged.

Security Service. We implement the security service in Python using PSY-TaLiRo [57] with the RTAMT backend [65] for dense-time STL robustness computation. Each of the 15 policies is defined as an STL specification and evaluated against the signal records from the instrumented application.

Algorithm 1 Click Signal Extraction**Input:** Click event e , cursor c , target o , scene S **Output:** Signal record R

```

1:  $R.valid \leftarrow EXISTS(c) \wedge ISAMEORIGIN(c)$ 
2:  $R.first\_intersected \leftarrow FIRSTHIT(c) = o$ 
3:  $ray \leftarrow CREATERAY(c.origin, c.direction)$ 
4:  $hits \leftarrow INTERSECT(ray, S.objects)$ 
5:  $k \leftarrow \text{index of } o \text{ in } hits$ 
    $\triangleright$  First visible object before and after  $o$  along the ray
6:  $R.visible\_infront \leftarrow FIRSTVISIBLE(hits, 0, k - 1)$ 
7:  $R.visible\_behind \leftarrow FIRSTVISIBLE(hits, k + 1, |hits| - 1)$ 
    $\triangleright$  Geometric overlap with any other hit
8:  $R.same\_shape\_space \leftarrow \exists x \in hits \setminus \{o\} : SAMESHAPESPACE(o, x)$ 
    $\triangleright$  Remaining signals
9:  $R.out\_fov \leftarrow ISOUTOFFOV(o, S.camera)$ 
10:  $R.will\_bubble \leftarrow e.bubbles \wedge \neg e.cancelled$ 
11: return  $R$ 

```

8 Evaluation

We evaluate EXRPOSE on a Meta Quest 3 headset across three studies. We show that EXRPOSE detects all ground-truth policy violations across 15 attacks in four WebXR applications. Its per-event overhead averages 3.85 ms, well within the 13.89 ms frame interval at 72 Hz. In a within-subjects user study with 20 participants, EXRPOSE detects all violations with no significant impact on presence or task load, and participants prefer alerts over being unaware of attacks. We address the following research questions.

- RQ1:** How effectively does EXRPOSE detect policy violations across attacks and applications?
- RQ2:** Is EXRPOSE’s runtime overhead low enough for real-time detection?
- RQ3:** Does EXRPOSE remain effective under real user interactions, and how does it affect user experience?

8.1 Experiment Setup

WebXR Applications. We use four WebXR applications (Gaming, Shopping, Travel, and Reading), representative of common XR use cases and used in prior work [34, 47]. Built with A-Frame [2] and Three.js [58], each features multiple interactive objects and 3D models, which we adapt to our experimental setup. The user locates and clicks target objects appearing at random positions to earn points (Gaming), browses and purchases clothes and accessories (Shopping), teleports between 360° views (Travel), and selects a book from a library, pages through it, and returns for another (Reading). The applications do not contain ads; we integrate the custom ad objects each experiment requires.

Attack Integration. We implement the 15 attacks in all four applications using existing implementations from prior work [14, 34, 47], adapting each to the application’s interaction context and yielding $15 \times 4 = 60$ attack scenarios. For example, in the DoS attack [14], a transparent ad overlays the target objects and intercepts the user’s clicks. Table 4 summarizes one representative integration per attack. Ad types vary by attack and include banner, video, YouTube, and iframe-served ads.

Device Setup. All experiments run on a Meta Quest 3 [42] headset: the application runs in the browser, the user enters an immersive

session via the VR button, and instrumented code captures timestamped signal logs on the device. For the effectiveness (§8.2) and performance (§8.3) experiments, the PSY-TaLiRo-based security service analyzes the logs post hoc on a laptop (Apple M4, 16 GB RAM), as no mature JavaScript library yet exists for in-browser STL monitoring. Since runtime alerts require only a binary outcome, the user study (§8.4) uses a JavaScript variant of the service running on the headset to display real-time alerts, which we verify against PSY-TaLiRo for consistency.

Data Collection and Validation. Two authors review the attack implementations and choose thresholds that separate attacks from ordinary UI behavior (e.g., opacity 0.1, below which overlays are effectively invisible), then independently interact with each scenario three times for one minute each, yielding $15 \times 4 \times 2 \times 3 = 360$ sessions. We also vary scene composition to measure overhead under diverse conditions and separately assess EXRPOSE with real users. We establish ground truth by instrumenting the application code to log each attack occurrence and comparing these logs against EXRPOSE’s detections. Where the user may legitimately click a partially visible ad (e.g., Visual Overlapping), we distinguish intentional from intercepted clicks by which object the click targets.

8.2 EXRPOSE’s Effectiveness (RQ1)

We measure detection effectiveness against ground truth and report results per attack, application, event type, and policy in Table 4.

Overall Correctness. Across 360 sessions, EXRPOSE observes 26,877 unique events, performs 29,868 policy checks, and correctly detects all 10,598 ground-truth violations with zero false negatives. Every violation produces a negative robustness value. We highlight cases that exercise distinct capabilities. In Malvertising, each click on the transparent layer in front of the visible target yields a P_1 violation (327 across all applications), and each leads to a cross-origin redirection (P_4) in the browser tab while the user remains in the 3D session, demonstrating detection across event types within a single attack. In Sequential Rendering (reading application), an ad loads behind a transparent layer in front of the return-to-library button, hiding it (P_8); when the user later clicks the button, the invisible ad receives the click while emitting a programmatic click to the button (P_2), demonstrating detection of an attack whose stages unfold at different times (47 violations of each policy).

In DoS through Overriding, 661 of 678 clicks stem from invisible controller cursors (P_6^{click}), and 83 of 122 sampled focus events show the inactive gaze cursor focusing on objects it cannot select (P_{12}); the remaining 17 clicks, generated by the gaze cursor before the attack overrides it, are correctly not flagged, demonstrating that EXRPOSE distinguishes the benign phase from the attack phase within a session. In Visual Overlapping (Travel), one of the 66 clicks landed directly on the ad, with no first-party object in front of it; P_5 correctly did not fire (65 violations), while P_2 still caught the synthetic click forwarded to the teleportation target (66), demonstrating that detection follows the observed geometry rather than the ad’s mere presence. Beyond binary detection, the robustness magnitude measures how far each signal lies from the violation boundary, letting providers prioritize violations or calibrate thresholds; Table 5 illustrates this for one policy per STL pattern (§6.1), separating clear violations from borderline cases.

Table 4: 15 UI-based attacks, each integrated into all four WebXR apps, with per-attack detection results. One representative integration and its triggered policies are shown per attack; the full per-app set is in our artifact [45]. $n = 24$ one-minute sessions per attack ($4 \text{ apps} \times 2 \text{ authors} \times 3 \text{ sessions}$); detections are reported per policy and app.

UI Attack	Implementation (Representative Scene)	Event Type	Policy	# Events	# Policy Checks	# Detected / # Ground Truth				Avg. Log (ms)	Avg. Verify (ms)
						Gaming	Shopping	Reading	Travel		
Abuse of Auxiliary Display [34]	On VR entry (Travel), injects an ad iframe onto the 2D mirror display (P_{10}) and opens a cross-origin page (P_7), both unseen from 3D mode.	redirect iframe_create	P_7	24	24	6/6	6/6	6/6	6/6	0.07 ± 0.08	0.34 ± 0.02
			P_{10}	24	24	6/6	6/6	6/6	6/6	0.17 ± 0.13	0.37 ± 0.04
Blind Spot Tracking [34]	An ad (Gaming) tracks the user's gaze, repositioning behind the head each frame to stay out of view past the allowed duration (P_{15}).	blind_spot	P_{15}	24	24	6/6	6/6	6/6	6/6	0.18 ± 0.14	0.50 ± 0.10
Controller Cursor-Jacking [34]	A hidden 180°-mirrored controller clicks an out-of-FoV rear ad whenever the user clicks a comic cover (Reading) (P_3).	click	P_3	2054	2054	61/61	43/43	410/410	105/105	2.15 ± 1.66	0.53 ± 0.17
Clickjacking [14]	An ad clones a product's shape and size and is created just before it at the same location (Shopping); the click resolves to the ad, not the product (P_9).	click	P_9	1600	1600	68/68	116/116	76/76	150/150	1.54 ± 1.51	0.43 ± 0.20
Denial-of-Service [14]	A near-transparent ad is overlaid on a product (Shopping); it catches the click, so the product never responds (P_1).	click	P_1	1737	1737	169/169	216/216	266/266	90/90	2.26 ± 1.69	0.76 ± 0.39
DoS through Overriding [47]	Teleportation targets are overridden to a raycast class the gaze cursor doesn't share, so gaze focus crosses classes and is denied (Travel) (P_{12}); only injected, near-invisible controllers can focus and click them (P_6).	click focus [‡] focus [§]	P_6^{click}	678	678	81/81	78/78	159/159	343/343	3.35 ± 1.34	0.34 ± 0.10
			P_6^{focus}	3096	3096	811/811	734/734	459/459	1024/1024	0.01 ± 0.03	0.34 ± 0.31
			P_{12}	122	122	27/27	12/12	17/17	27/27	1.74 ± 1.28	0.55 ± 0.07
Event Bubbling	An ad is made a DOM ancestor of a book page (Reading); a click on the page bubbles up to the ad (P_{13}), which was never the cursor's target (P_2).	click	P_{13}	1066	1066	21/21	53/53	204/204	81/81	2.51 ± 1.16	0.46 ± 0.38
			P_2		1066	21/21	53/53	204/204	81/81		0.59 ± 0.56
Gaze Cursor-Jacking [34]	The real gaze cursor is made invisible and offset while a visible decoy marks where the user aims; the hidden cursor clicks a nearby ad instead of the armadillo (Gaming) (P_6).	click focus [‡]	P_6^{click} P_6^{focus}	949 3050	949 3050	85/85 320/320	84/84 432/432	89/89 676/676	107/107 363/363	1.61 ± 1.33 0.01 ± 0.03	0.33 ± 0.04 0.34 ± 0.28
GUI Switch [47]	Periodic screenshots capture the view without a user click (P_{11}); on exit, a replica page replaces the tour via a cross-origin redirect (Travel) (P_7 , P_4).	replace	P_4	24	24	6/6	6/6	6/6	6/6	0.60 ± 0.85	0.35 ± 0.02
		screenshot	P_7	24	24	6/6	6/6	6/6	6/6		0.33 ± 0.02
			P_{11}	246	246	63/63	61/61	62/62	60/60	0.08 ± 0.32	0.32 ± 0.03
Input Forgery [14]	An ad follows behind the user, out of view (Shopping) (P_{15}); it receives forged synthetic clicks that fake user interaction (P_2).	click	P_2	920	920	49/49	53/53	48/48	47/47	2.23 ± 1.50	0.57 ± 0.41
		blind_spot	P_{15}	24	24	6/6	6/6	6/6	6/6	0.17 ± 0.15	0.46 ± 0.03
Intercepting User Inputs [14]	Transparent overlays cover each key of a PIN keypad (Reading), catching every typed digit (P_1); a synthetic click then submits the sniffed PIN (P_2).	click	P_1	925	925	42/42	40/40	41/41	34/34	3.37 ± 1.85	0.75 ± 0.55
			P_2		925	10/10	9/9	10/10	7/7		0.57 ± 0.36
Malvertising [47]	A near-transparent ad overlays a teleportation target (Travel); it intercepts the click (P_1) and opens a cross-origin page (P_4).	click redirect	P_1	902	902	101/101	50/50	106/106	70/70	2.51 ± 1.93	0.78 ± 0.65
			P_4	654	654	101/101	50/50	106/106	70/70	0.08 ± 0.11	0.32 ± 0.05
Object Erasure [14]	An in-view ad has its opacity mutated to near-zero (Gaming), erasing it from view while it stays present in the FoV (P_{14}).	material_change	P_{14}	1316	1316	43/43	41/41	35/35	30/30	0.10 ± 0.09	0.75 ± 0.31
Sequential Rendering [47]	A transparent, unclickable first-party cover renders first; an ad is then created behind it (Gaming). Loaded later, the ad is obscured (P_8) yet still takes the click and forwards a synthetic click to the visible restart button behind (P_2).	click	P_2	957	957	6/6	33/33	47/47	48/48	2.47 ± 1.63	0.62 ± 0.86
		obj_create/pos [¶]	P_8	5509	5509	6/6	33/33	47/47	48/48	2.66 ± 2.41	1.17 ± 0.31
Visual Overlap-ping [47]	An ad sits behind a visible, non-clickable pay button (Shopping); the click passes through to the ad (P_5), which forwards a synthetic click to the pay button (P_2).	click	P_2	976	976	6/6	45/45	49/49	66/66	2.72 ± 2.21	0.55 ± 0.05
			P_5		976	6/6	45/45	49/49	65/65		0.43 ± 0.05
Overall	15 UI-based attacks $\times$ 4 apps	—	P_1 - P_{15}	26,877	29,868	2,133/2,133	2,317/2,317	3,196/3,196	2,952/2,952	1.67 ± 1.97	0.63 ± 0.46

All ads are third-party content. [‡] Focus listener event, [§] Focus monitor (periodic sampling). [¶] obj_create/pos = obj_create, pos_change.

Table 5: Illustrative robustness scores for selected policies.

Policy	Varying Factor	Threshold	Robustness	Outcome
P_1	opacity = 0.05, cross-party behind	$\theta = 0.1$	-0.05	Violation
	opacity = 0.15, cross-party behind		+0.05	Safe
	opacity = 0.05, no object behind		+0.50	Safe
P_7	redirect 200 ms after scene entry	$T = 1000 \text{ ms}$	-800	Violation
	redirect 1200 ms after scene entry		+200	Safe
P_{11}	screenshot 30 ms after click	$T = 100 \text{ ms}$	+70	Safe
	screenshot 150 ms after click		-50	Violation

8.3 Performance Overhead (RQ2)

We measure whether eXRPOSE's overhead fits within the time available per rendered frame. WebXR frameworks render at 72 Hz, so we report overhead against the 13.89 ms frame interval and decompose it into logging time (signal extraction per event) and verification

time (policy evaluation per check). We first report overhead in the attack scenarios, then characterize how logging and verification scale with scene composition, validate the results on production WebXR applications, and report the one-time instrumentation cost.

8.3.1 Overhead in Attack Scenarios. Across all 26,877 events in Table 4, logging adds $1.67 \pm 1.97 \text{ ms}$ per event and policy evaluation adds $0.63 \pm 0.46 \text{ ms}$ per check across 29,868 checks. Since an event may trigger multiple policies (seven for click), the combined per-event overhead averages $3.85 \pm 2.43 \text{ ms}$ (27.7% of the frame interval), leaving substantial headroom for the application's own rendering and interaction logic. Policy checks occur only at interaction events, not every frame, so the amortized overhead is 0.39% at one event per second.

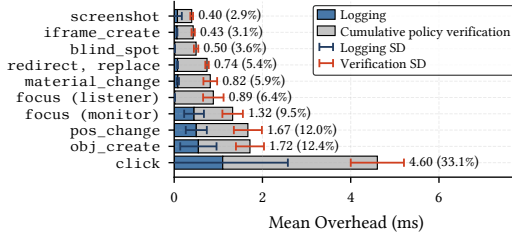


Figure 5: End-to-end runtime overhead per event type on Meta Quest 3 (72 Hz, 13.89 ms frame budget).

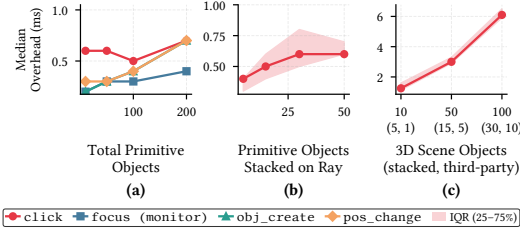


Figure 6: Median logging overhead with scene complexity. (a) Logging overhead vs. total primitive objects. (b) Click overhead vs. primitive objects along the cursor ray. (c) Click overhead vs. combined scene complexity with 3D objects.

8.3.2 Scaling Analyses. Logging cost depends on scene complexity, so we measure it across scene compositions varying the total object count (10–200), objects along the cursor’s ray (5–50), third-party proportion (0–20), and geometry type (primitives vs. 3D models), with 50–300 samples per scene (full configurations in artifact [45]). Verification cost depends only on the number and type of policies evaluated, so we report it from the attack scenarios together with a controlled benchmark varying policy count per click. Because per-frame timings are right-skewed by occasional garbage-collection and scheduler spikes, the scaling analyses report medians with interquartile ranges, while per-event and per-policy costs use means, where total consumed frame time is the quantity of interest.

End-to-End Overhead. Figure 5 shows the total per-event overhead. Click events cost the most at 4.60 ms (33.1% of the frame interval), driven by scene-dependent raycasting and seven policies per click. Object creation (1.72 ms), position change (1.67 ms), and focus monitoring (1.32 ms) follow, and all remaining event types stay under 1 ms. No event type exceeds the frame budget, and clicks occur at most 2–3 times per second, so their amortized cost stays under 2% of the frame stream.

Logging Overhead. Figure 6 shows how median logging overhead scales with scene complexity. Constant-time event types (focus listener, material change, blind spot, redirection, iframe creation, screenshot) perform single-object property lookups and remain under 0.1 ms regardless of scene complexity. Three event types scale with scene density. The focus monitor grows from 0.20 to 0.40 ms, object-creation logging from 0.20 to 0.70 ms, and position logging from 0.30 to 0.70 ms as objects increase from 10 to 200 (Figure 6a). Click logging is dominated by objects along the cursor’s ray, since

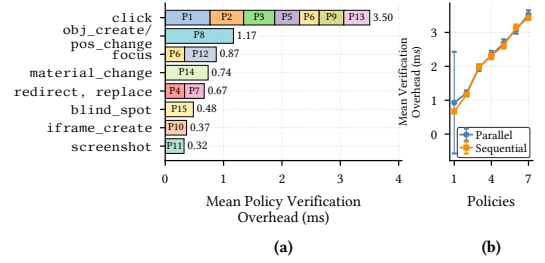


Figure 7: Mean verification overhead analysis. (a) Verification overhead per event type with per-policy breakdown. (b) Verification overhead vs. number of policies (1–7).

Table 6: Click logging overhead vs. scene complexity.

	App	Total Objects	3D-mesh Objects	Max on Ray	Click Logging Overhead (ms)
Study apps [34, 47]	Reading	64	0	5	1.39
	Travel	86	0	2	2.76
	Shopping	114	0	6	3.28
	Gaming	37	21	6	3.24
Production apps	a-blast [1]	162	95	5	5.25
	a-saturday [5]	66	27	5	6.02
	hallovreen [27]	34	15	9	9.45

each intersected object requires opacity and bounding-box checks, while off-ray objects add only a cheap bounding-sphere rejection. Click overhead is therefore largely insensitive to total object count but scales with ray depth, from 0.40 ms at depth 5 to 0.60 ms at depth 50 (Figure 6b); the third-party ratio has little effect. With 3D models, overhead grows sharply as mesh hierarchies require deeper inspection (Figure 6c): a medium scene reaches 3.0 ms and the worst case (100 objects, 30 along the ray) reaches 6.1 ms (44% of the frame interval), far above any primitive scene (≤ 0.8 ms).

Verification Overhead. Figure 7 summarizes mean verification overhead. Click events cost 3.50 ms across their seven policies, with individual policies ranging from 0.34 ms (P_6) to 0.77 ms (P_1); single-policy events range from 0.32 ms to 1.17 ms, and two-policy events cost 0.67–0.87 ms (Figure 7a). Verification scales linearly at approximately 0.43 ms per additional policy, with sequential and parallel evaluation performing equivalently over 1–7 policies (seven being the most any single event triggers) (Figure 7b).

8.3.3 Real-World Overhead. To test generalization beyond our four applications, we measure click logging overhead on three production WebXR applications, *a-blast* [1], *a-saturday-night* [5], and *hallovreen* [27], with EXRPOSE integrated unmodified. We focus on clicks because the handler raycasts the entire scene, making it the costliest and most scene-dependent event type; constant-time types are unaffected by scene composition. Table 6 pairs each application’s scene complexity (total objects, 3D-mesh objects, and maximum objects along the cursor ray) with its click overhead. Applications built entirely from primitives (Reading, Travel, Shopping) load no 3D meshes and are the cheapest at 1.4–3.3 ms. Mesh-based applications range from Gaming (21 meshes, 3.24 ms) to the production applications (5.25–9.45 ms), but mesh count alone does not rank them. *a-blast* loads the most meshes (95 low-poly models) yet costs only 5.25 ms, whereas *hallovreen*’s 15 detailed models and deeper

rays (9 objects intersected) yield the highest overhead, 9.45 ms. Even this heaviest application stays within the frame budget (68%), consistent with our controlled analysis, where meshes and ray depth rather than raw object count drive click cost (Figure 6c).

8.3.4 Instrumentation Cost. The instrumentor is an approximately 1,700-line external module requiring ~15 added lines in the application source; the automated insertion script finishes in under one second. Instrumentation occurs before execution and adds no runtime overhead, and all components initialize in under 1.5 ms combined on the Meta Quest 3.

8.4 User Experience and Interaction (RQ3)

We evaluate whether EXRPOSE remains robust under real-user interactions and whether its enforcement affects user experience.

8.4.1 Study Design. We conducted an IRB-approved, within-subjects, in-person user study with $N = 20$ participants. Each participant experienced our study gaming application [61] under two conditions, *with-EXRPOSE* (policy violations trigger warnings) and *without-EXRPOSE* (no monitoring), playing two one-minute sessions per condition. We selected the gaming application because sessions are strictly timed, the task (locating and clicking objects) needs no guidance, and its 3D models add realistic scene complexity. We integrated four attacks into the scene (Event Bubbling, Malvertising, Denial-of-Service, and Visual Overlapping), triggering five policy violations during gameplay (P_1 , P_2 , P_4 , P_5 , and P_{13}). When a violation is detected in the EXRPOSE condition, a descriptive warning appears on a board within the 3D scene (Figure 8).

Both conditions contained the same attacks in the same order but used different ad to mitigate learning effects; condition order was counterbalanced across participants (10 per order), and we did not disclose information about ads, warnings, or unintended interactions at the start. A pilot with two participants led to three changes. We used different ads across conditions, highlighted key phrases in the warning text, and displayed a warning per violation rather than a single aggregate notification, so warning frequency reflects violation frequency.

Survey and Interview. After each condition, participants completed a Qualtrics survey measuring two constructs on 7-point scales (−3 to +3), comprising three presence items adapted from the IPQ [26] (e.g., anchored fully disagree to fully agree) and four task-load items adapted from the NASA-TLX [48] (e.g., anchored very low to very high), along with open-ended questions capturing immediate reactions. After both conditions, we conducted a 12–15 minute semi-structured interview covering open recall, threat comprehension, reaction and behavioral intent, design critique of the warnings, and closing remarks. The interview also served as the deception debrief; we began without revealing the attacks and disclosed them during the reaction phase, capturing both unprompted and informed responses, including whether participants would behave differently on their own device. The survey questionnaire and interview protocol are in our artifact repository [45].

Recruitment. We recruited 20 participants from our university through an IRB-approved procedure; a pre-screening survey excluded medical conditions that could cause adverse VR effects (e.g., photosensitive epilepsy), and none were reported. Sessions

Table 7: Participant demographics ($N = 20$).

Category	Distribution
Age	18–24: 8; 25–34: 11; 35–44: 1
Gender	Male: 13; Female: 6; Prefer not to say: 1
Race/Ethnicity	Asian: 17; White: 3
Education	High school diploma: 1; Bachelor's: 6; Master's: 2; Doctoral: 11
Programming skills	Yes: 20
VR experience	Yes: 19; No: 1
VR usage	Gaming: 13; AR/VR/MR development: 2; Other (tours, 360° video, etc.): 8

Table 8: Presence and workload.

Item	Without EXRPOSE	With EXRPOSE	Wilcoxon p	Effect (r)
Felt present	2.40 ± 0.68	2.25 ± 0.72	0.257	0.038
Realism	1.20 ± 1.44	1.20 ± 1.47	1.000	0.050
Attention to real env	−1.55 ± 1.76	−1.55 ± 1.43	0.963	0.155
Mental demand	−0.80 ± 1.85	−0.75 ± 1.68	0.755	0.095
Success	2.00 ± 0.86	1.60 ± 1.35	0.161	0.029
Effort	−0.20 ± 1.67	−1.05 ± 1.64	0.018*	0.112
Frustration	−1.65 ± 1.69	−1.50 ± 1.82	0.586	0.129

* Significant result. 7-point Likert scales, −3 to +3; anchors vary by item.

averaged 43 minutes (maximum 50), and participants received a \$20 gift card. Table 7 summarizes participant demographics, and Appendix Section C discusses risk mitigation.

Data Analysis. Because the dependent measures are ordinal Likert items (−3 to +3), we compared conditions per item using paired Wilcoxon signed-rank tests [64], reporting effect sizes as r with significance at $\alpha = 0.05$. To check for order effects, we compared the per-participant difference (with – without) between the two order groups using Mann–Whitney U tests [38]. For qualitative analysis, we applied an inductive-deductive approach; transcripts were double coded, with disagreements resolved in meetings and high pre-resolution agreement (Cohen's κ [15] > 0.8).

8.4.2 Results. We report the impact on presence and task load, detection correctness, and qualitative findings.

Presence, Task Load, and Order Effect. Table 8 summarizes the presence (rows 1–3) and task load (rows 4–7) scores across conditions. None of the three IPQ presence items differed significantly between conditions, indicating that EXRPOSE did not diminish immersion, and three of four NASA-TLX items also showed no significant difference. The exception was self-reported effort, significantly lower in the EXRPOSE condition ($p = 0.018$). We hypothesize that warnings reduced the need to independently reason about unresponsive clicks; without EXRPOSE, participants repeatedly clicked targets without understanding why their actions failed, perceiving this as greater effort. Mann–Whitney U tests on the per-participant differences revealed no significant order effects, supporting pooling across condition orders.

Effectiveness Under Interaction Variability. Table 9 presents detection results across all 20 participants, with logging measured on the Quest 3 and verification computed post hoc with PSY-TaLiRo. EXRPOSE detected all policy violations, with click logging (4.17 ms) within the gaming application's 1.9–4.3 ms per-attack range (RQ1) and well within the frame budget. During the study, warnings were displayed by the in-app JavaScript verifier, which completes verification under 0.4 ms for click-triggered policies. Warning display latency after detection ranged from 15.72 ± 1.81 ms (P_2) to

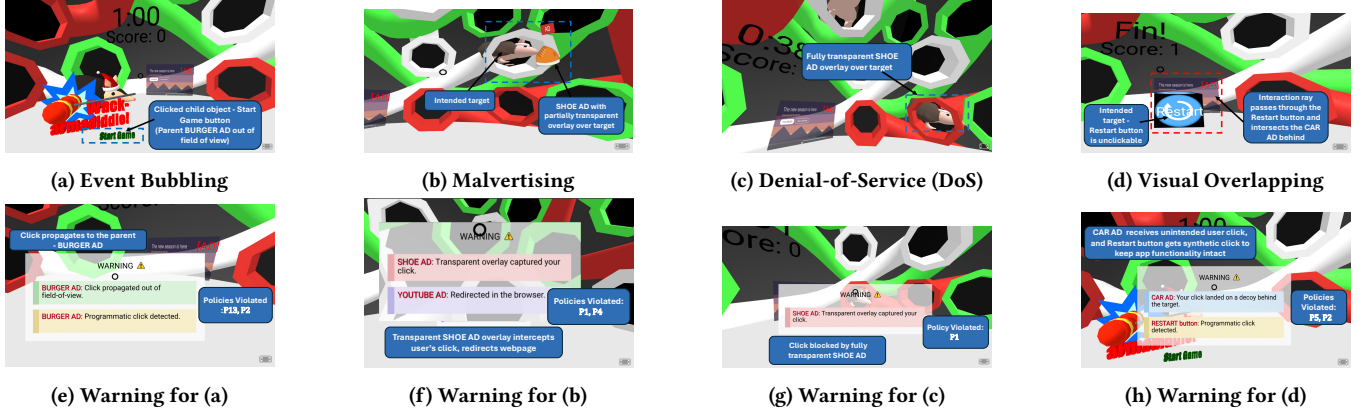


Figure 8: Top row: four UI-based attacks in the user-study gaming scene. Bottom row: corresponding EXRPOSE warnings. Scene adapted from Wackarmadiddle [61] (MIT License, © 2018 Srile). Includes the Armadillo model by Google, originally distributed via Google Poly under CC BY 3.0. The Santa Hat model is by Joe Dorman, also via Google Poly under CC BY 3.0.

Table 9: User study detection results ($N = 20$).

Event Type	Condition	Policy	# Events/User	# Ground Truth/User	Detection Rate	Avg. Log (ms)	Avg. Verify (ms)
click	with EXRPOSE	P_1	77.05 ± 20.34	34.7 ± 17.17	100%	4.17 ± 0.29	0.71 ± 0.09
		P_2		3.0 ± 0.00	100%		0.55 ± 0.10
		P_5		0.95 ± 0.22	100%		0.44 ± 0.08
		P_{13}		2.0 ± 0.00	100%		0.40 ± 0.03
	without EXRPOSE	P_1	115.05 ± 45.70	68.2 ± 47.06	—	—	—
		P_2		3.0 ± 0.00	—	—	—
redirect	with EXRPOSE	P_4	2.0 ± 0.00	1.0 ± 0.00	100%	0.12 ± 0.13	0.37 ± 0.03
		P_5		0.95 ± 0.22	—	—	—
	without EXRPOSE	P_4	2.0 ± 0.00	1.0 ± 0.00	—	—	—
		P_5		0.95 ± 0.22	—	—	—

32.37 ± 2.95 ms (P_5), dominated by DOM rendering rather than policy evaluation. Notably, P_1 violations were substantially fewer with EXRPOSE (34.7 ± 17.2 per user) than without (68.2 ± 47.1), suggesting warnings reduced repeated clicks on the transparent overlay.

Qualitative Findings. Our qualitative analysis yielded 63 codes organized into three themes covering experience differences between conditions, the shift in awareness and behavioral intent after disclosure, and design recommendations.

Experience differences between conditions. No participant suspected malicious interference in the without-EXRPOSE condition. 60% reported a smooth experience, and click failure was the primary observable symptom (70%), attributed to self-error (50%) or software/hardware issues (30%): “I wasn’t standing at the correct distance, so some of my clicks did not go properly... I have to, like, double press” (U17). In contrast, 70% reported that the warnings in the EXRPOSE condition led them to reinterpret these click failures (warning-induced awareness): “Because of all the warnings, I realized how many ads there were that I couldn’t see... on the first task, it was like, whoa, maybe there’s tons of ads I’ve been clicking on, or bad things happening, and I just have no clue... I couldn’t avoid it” (U1). However, 45% misclassified the warnings as system errors, 80% focused primarily on the game (task-focus), and 35% did not read the warnings carefully (warning-neglect). Despite this, 85%

wanted to investigate the warnings by clicking, reading, or looking around (investigation-intent).

After disclosure, informed preference for detection. After learning about the attacks, all 20 participants expressed a preference for detection, and 95% stated their behavior would differ in a real-world scenario (real-world behavior change), noting they focused on gameplay because they knew nothing harmful would occur. The most common intent was to stop or abandon the application (65%): “If it was on my system, I would have immediately quit and checked” (U6). 20% cited financial data or credentials as their primary concern: “I don’t want to be visiting unintended websites, or leading to some sort of leakage of financial data... I would 100% want to be informed” (U7). 25% requested pre-click prevention, and 10% would pause and inspect their browser, accounts, or system for damage.

Design recommendations. The most frequent request was for the experience to pause on warning (75%): “If it paused... without the game currently happening around me, I’d have more time to think about it” (U2). 45% requested allow/block controls modeled on browser dialogs, 45% suggested peripheral placement, and 20% suggested severity-dependent warnings. Participants also recommended specific threat details (45%), color-coded icons (40%), and consolidated warnings instead of duplicates (25%). 15% suggested the platform should handle detection automatically (automatic-handling), reducing user burden: “This kind of application seems very sophisticated, so it should be controlled by a developer... inside the software stack... it should do all that stuff automatically” (U8).

9 Discussion and Limitations

Application Context. Our evaluation reflects a core WebXR challenge, as the same UI properties can support both benign functionality and malicious behavior. A synthetic click may legitimately trigger a distant object in an animation, and a transparent object in front of a target may intercept clicks maliciously or serve as a benign interaction boundary, collision region, or visual-effect layer. Distinguishing the two requires knowing what is typical in legitimate WebXR applications, and EXRPOSE’s policies can be extended as such norms emerge from future measurement studies.

Generalizability. EXRPOSE is implemented for A-Frame and Three.js, but its policies and predicates are defined over concepts common to any 3D scene framework, such as opacity, spatial ordering, and field of view; porting requires reimplementing only the code instrumentor for the target framework's event and scene-graph APIs. For future attacks, our policy construction applies in the same way, identifying the unintended interaction pattern, selecting events and predicates, and formalizing the violation in STL.

Deployment Considerations. We assume EXRPOSE is integrated by trusted platform providers such as browsers, WebXR frameworks, or operating systems. Our user study demonstrates feasibility with an in-app verifier that alerts users in real time; platform-level deployment across applications, devices, and enforcement modes requires further testing. In our study applications, EXRPOSE's end-to-end overhead remains under 5 ms per event, well below the 63–130 ms latencies reported to degrade VR user experience [12, 28]. Repeated checks, though individually cheap, can accumulate overhead, so deployments should rate-limit them. We showed alerts without blocking so participants could interact freely; they preferred warnings over remaining unaware and wanted a pause on severe events. Prior HCI work on XR notifications [19, 23, 33, 35, 41, 49, 54] can guide warning design toward alerts that are noticeable without disrupting interaction.

10 Related Work

UI Attacks and Defenses in Other Domains. Prior work in web, mobile, and IoT systems shows that user perception and interaction can be manipulated through clickjacking, overlay-based deception, and dark patterns [6, 10, 18, 24, 30, 39, 53, 67]. Defenses include mobile clickjacking protections, automated web detection, and policy-based mitigation for smart environments [9, 11, 13, 51]. However, they target 2D interfaces and do not address the spatial composition, multiple input modalities, and visibility manipulation of immersive environments.

UI Attacks and Defenses in XR. Recent work identifies security-sensitive UI properties in immersive scenes (e.g., same-space placement, opacity manipulation, synthetic input) that enable click fraud, impression fraud, functionality disruption, and data extraction [14, 20, 32, 34, 44, 46, 47, 56, 60, 62]. Defenses remain limited to sand-boxing of third-party content [34], automated application analysis [29], provenance-based investigation [55], and runtime scene scanning [36], which target narrow settings, rely on trusted developer configuration, or perform offline analysis rather than runtime enforcement. In contrast, EXRPOSE derives runtime-enforceable policies from known UI attacks, characterizes unintended interactions through contextual information, and enforces them online.

11 Conclusion

We introduce EXRPOSE, a runtime detection system for UI-based attacks in WebXR that characterizes unintended interactions as events and predicates, formalizes them in Signal Temporal Logic, and evaluates runtime signals against the resulting policies. On a Meta Quest 3, EXRPOSE detects all violations across 15 attacks in four applications within the frame interval, and a 20-participant user study confirms detection under real interactions with no significant impact on presence or task load, with participants preferring

alerts over unawareness. This work is a step toward safe adoption of immersive web environments.

12 Acknowledgments

We appreciate the valuable feedback and suggestions from our anonymous reviewers. This research was partially funded by the National Science Foundation (NSF) through grants CNS-2144645, CNS-2544561, and IIS-2229876. The findings, conclusions, and recommendations presented in this paper are solely those of the authors and do not necessarily represent the views of the NSF.

References

- [1] A-Blast <https://github.com/aframevr/a-blast>. [Accessed: 06-Sep-2026].
- [2] A-Frame - web framework for building virtual reality experiences <https://aframe.io/>. [Accessed: 06-Sep-2026].
- [3] A-Frame showcase hello world <https://aframe.io/examples/showcase/helloworld/>. [Accessed: 06-Sep-2026].
- [4] A-Painter <https://aframe.io/a-painter/>. [Accessed: 06-Sep-2026].
- [5] A-Saturday-Night <https://github.com/aframevr/a-saturday-night>. [Accessed: 06-Sep-2026].
- [6] Devdatta Akhawe, Warren He, Zhiwei Li, Reza Moazzezi, and Dawn Song. 2014. Clickjacking Revisited: A Perceptual View of UI Security. In *USENIX Workshop on Offensive Technologies*.
- [7] Apple Vision Pro <https://www.apple.com/apple-vision-pro/>. [Accessed: 06-Sep-2026].
- [8] Babylon.js <https://www.babylonjs.com/>. [Accessed: 06-Sep-2026].
- [9] Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, and Christopher Kruegel. 2010. A Solution for the Automated Detection of Clickjacking Attacks. In *ACM Symposium on Information, Computer and Communications Security*.
- [10] Philipp Beer, Marco Squarcina, Sebastian Roth, and Martina Lindorfer. 2025. TapTrap: Animation-Driven Tapjacking on Android. In *USENIX Security*.
- [11] Davide Bove. 2022. SoK: The Evolution of Trusted UI on Mobile. In *ACM on Asia Conference on Computer and Communications Security*.
- [12] Polona Caserman, Michelle Martinussen, and Stefan Göbel. 2019. Effects of End-to-end Latency on User Experience and Performance in Immersive Virtual Reality Applications. In *Joint International Conference on Entertainment Computing and Serious Games*.
- [13] Z. Berkay Celik, Gang Tan, and Patrick McDaniel. 2019. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. In *NDSS*.
- [14] 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*.
- [15] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*.
- [16] Content Security Policy Level 3 <https://www.w3.org/TR/CSP3/>. [Accessed: 06-Sep-2026].
- [17] ECMAScript Language Specification 2023. <https://262.ecma-international.org/14.0/>. [Accessed: 06-Sep-2026].
- [18] Yanick Fratantonio, Chenxiong Qian, Simon P. Chung, and Wenke Lee. 2017. Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop. In *IEEE Symposium on Security and Privacy (S&P)*.
- [19] Sarthak Ghosh, Lauren Winston, Nishant Panchal, Philippe Kimura-Thollander, Jeff Hotnag, Douglas Cheong, Gabriel Reyes, and Gregory D. Abowd. 2018. NotificationVR: Exploring Interruptions and Notifications in Virtual Reality. *IEEE Transactions on Visualization and Computer Graphics*.
- [20] Hilda Hadan, Lydia Choong, Leah Zhang-Kennedy, and Lennart E. Nacke. 2024. Deceived by Immersion: A Systematic Analysis of Deceptive Design in Extended Reality. *ACM Comput. Surv.*
- [21] heyVR Aframe Advertisement SDK <https://docs.heyvr.io/game-development/sdk/advertisement/aframe>. [Accessed: 06-Sep-2026].
- [22] HoloLens 2 <https://learn.microsoft.com/en-us/hololens/hololens2-hardware>. [Accessed: 06-Sep-2026].
- [23] Ching-Yu Hsieh, Yi-Shyuan Chiang, Hung-Yu Chiu, and Yung-Ju Chang. 2020. Bridging the Virtual and Real Worlds: A Preliminary Study of Messaging Notifications in Virtual Reality. In *CHI Conference on Human Factors in Computing Systems*.
- [24] Lin-Shung Huang, Alex Moshchuk, Helen J. Wang, Stuart Schecter, and Collin Jackson. 2012. Clickjacking: Attacks and Defenses. In *USENIX Security*.
- [25] <iframe>: HTML inline frame element <https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/iframe>. [Accessed: 06-Sep-2026].
- [26] Igroup Presence Questionnaire (IPQ) <https://www.igroup.org/pq/ipq/>. [Accessed: 06-Sep-2026].

- [27] Jorge Fuentes. A Halloween webVR experience for kids. <https://github.com/zomboh/halloween>. [Accessed: 06-Sep-2026].
- [28] Viktor Kelkkanen, David Lindero, Markus Fiedler, and Hans-Jürgen Zepernick. 2023. Hand-Controller Latency and Aiming Accuracy in 6-DOF VR. *Advances in Human-Computer Interaction*.
- [29] John Y. Kim, Chaoshun Zuo, Yanjie Zhao, and Zhiqiang Lin. 2025. AUTOVR: Automated UI Exploration for Detecting Sensitive Data Flow Exposures in Virtual Reality Apps. In *USENIX Security*.
- [30] Monica Kowalczyk, Johanna T. Gunawan, David Choffnes, Daniel J Dubois, Woodrow Hartzog, and Christo Wilson. 2023. Understanding Dark Patterns in Home IoT Devices. In *CHI Conference on Human Factors in Computing Systems*.
- [31] Ron Koymans. 1990. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Syst.*
- [32] Veronika Krauß, Pejman Saeghe, Alexander Boden, Mohamed Khamis, Mark McGill, Jan Gugenheimer, and Michael Nebeling. 2024. What Makes XR Dark? Examining Emerging Dark Patterns in Augmented and Virtual Reality through Expert Co-Design. *ACM Trans. Comput.-Hum. Interact.*
- [33] May Jorella Lazaro, Sungho Kim, Jaeyong Lee, Jaemin Chun, and Myung-Hwan Yun. 2021. Interaction Modalities for Notification Signals in Augmented Reality. In *International Conference on Multimodal Interaction*.
- [34] Hyunjo Lee, Jiyeon Lee, Daejun Kim, Suman Jana, Insik Shin, and Soeul Son. 2021. AdCube: WebVR Ad Fraud and Practical Confinement of Third-Party Ads. In *USENIX Security*.
- [35] Hyunjin Lee and Woontack Woo. 2023. Exploring the Effects of Augmented Reality Notification Type and Placement in AR HMD while Walking. In *IEEE Conference Virtual Reality and 3D User Interfaces (VR)*.
- [36] Bertram Liu, Vamsi Shankar Simhadri, and Xiaokuan Zhang. 2025. Hunting Insecure UI Properties in Extended Reality. In *ACM MobiHoc XR-Security Workshop*.
- [37] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*.
- [38] Henry B. Mann and Donald R. Whitney. 1947. On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*.
- [39] Arunesh Mathur, Gunes Acar, Michael J. Friedman, Eli Lucherini, Jonathan Mayer, Marshini Chetty, and Arvind Narayanan. 2019. Dark Patterns at Scale: Findings from a Crawl of 11K Shopping Websites. *ACM Hum.-Comput. Interact.*
- [40] Arunesh Mathur, Mihir Kshirsagar, and Jonathan Mayer. 2021. What Makes a Dark Pattern... Dark? Design Attributes, Normative Considerations, and Measurement Methods. In *CHI Conference on Human Factors in Computing Systems*.
- [41] Andrea Mengascini, Rebecca Weil, Annabelle Walle, Jürgen Steimle, and Giancarlo Pellegrino. 2025. Exploring the Design Space for Security Warnings in Immersive Environments. In *IEEE European Symposium on Security and Privacy*.
- [42] Meta Quest 3 <https://www.meta.com/quest/quest-3/>. [Accessed: 06-Sep-2026].
- [43] Moon Rider <https://moonrider.xyz/>. [Accessed: 06-Sep-2026].
- [44] Chandrika Mukherjee, Reham Mohamed Aburas, Arjun Arunasalam, Habiba Farrukh, and Z. Berkay Celik. 2025. Demo: UI Based Attacks in WebXR. In *International Conference on Mobile Systems, Applications and Services*.
- [45] Chandrika Mukherjee, Aishwarya Devi Akila Pandian, Arjun Arunasalam, Jiasi Chen, Z. Berkay Celik, and Habiba Farrukh. 2026. EXRPOSE: Runtime UI-based Attack Detection in WebXR - Research Artifacts. <https://doi.org/10.6084/m9.figshare.33472219>.
- [46] Chandrika Mukherjee, Arjun Arunasalam, Habiba Farrukh, Reham Mohamed Aburas, and Z. Berkay Celik. 2025. Towards Secure User Interaction in WebXR. In *International Workshop on Human-Centered Sensing, Modeling, and Intelligent Systems*.
- [47] 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*.
- [48] NASA Task Load Index (TLX) <https://www.nasa.gov/human-systems-integration-division/nasa-task-load-index-tlx/>. [Accessed: 06-Sep-2026].
- [49] Lucas Plabst, Aditya Raikwar, Sebastian Oberdörfer, Francisco Raul Ortega, and Florian Niebling. 2023. Exploring Unimodal Notification Interaction and Display Methods in Augmented Reality. In *ACM Symposium on Virtual Reality Software and Technology*.
- [50] Amir Pnueli. 1977. The Temporal Logic of Programs. In *Symposium on Foundations of Computer Science*.
- [51] Andrea Possemato, Andrea Lanzi, Simon Pak Ho Chung, Wenke Lee, and Yanick Fratantonio. 2018. ClickShield: Are You Hiding Something? Towards Eradicating Clickjacking on Android. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.
- [52] David Ross and Tobias Gondrom. HTTP Header Field X-Frame-Options. <https://www.rfc-editor.org/rfc/rfc7034> [Accessed: 06-Sep-2026].
- [53] Gustav Rydstedt, Elie Bursztain, Dan Boneh, and Collin Jackson. 2010. Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites. *Web 2.0 Security and Privacy*.
- [54] Rufat Rzayev, Sven Mayer, Christian Krauter, and Niels Henze. 2019. Notification in VR: The Effect of Notification Placement, Task and Environment. In *Symposium on Computer-Human Interaction in Play*.
- [55] Muhammad Shoaib, Alex Suh, and Wajih UI Hassan. 2025. Principled and Automated Approach for Investigating AR/VR Attacks. In *USENIX Security*.
- [56] Ali Teymourian, Andrew M. Webb, Taha Gharaibeh, Arushi Ghildiyal, and Ibrahim Baggili. 2025. SoK: Come Together – Unifying Security, Information Theory, and Cognition for a Mixed Reality Deception Attack Ontology & Analysis Framework. In *USENIX Security*.
- [57] Quinn Thibeault, Jacob Anderson, Aniruddh Chandratre, Giulia Pedrielli, and Georgios Fainekos. 2021. PSY-TaLiRo: A Python Toolbox for Search-Based Test Generation for Cyber-Physical Systems. In *Formal Methods for Industrial Critical Systems*.
- [58] Three.js – JavaScript 3D Library <https://threejs.org/>. [Accessed: 06-Sep-2026].
- [59] Tower Max Fitness <https://towermax.fitness/tower/>. [Accessed: 06-Sep-2026].
- [60] Wen-Jie Tseng, Elise Bonnal, Mark McGill, Mohamed Khamis, Eric Lecolinet, Samuel Huron, and Jan Gugenheimer. 2022. The Dark Side of Perceptual Manipulations in Virtual Reality. In *CHI Conference on Human Factors in Computing Systems*.
- [61] Wackarmadiddle: Wack-a-mole with Armadillos and pipes in A-Frame <https://github.com/WhiteRabbit/wackarmadiddle>. [Accessed: 06-Sep-2026].
- [62] Xian Wang, Lik-Hang Lee, Carlos Bermejo Fernandez, and Pan Hui. 2024. The Dark Side of Augmented Reality: Exploring Manipulative Designs in AR. *International Journal of Human-Computer Interaction*.
- [63] WebXR Device API <https://www.w3.org/TR/webxr/>. [Accessed: 06-Sep-2026].
- [64] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*.
- [65] Tomoya Yamaguchi, Bardh Hoxha, and Dejan Nicković. 2024. RTAMT – Runtime Robustness Monitors with Application to CPS and Robotics. *International Journal on Software Tools for Technology Transfer*.
- [66] Zesty Advertisement SDK <https://github.com/zestyxyz/zesty-ads-sdk-web>. [Accessed: 06-Sep-2026].
- [67] Mingxue Zhang, Wei Meng, Sangho Lee, Byoungyoung Lee, and Xinyu Xing. 2019. All Your Clicks Belong to Me: Investigating Click Interception on the Web. In *USENIX Security*.

A Generative AI Usage

We used OpenAI ChatGPT and Anthropic Claude to improve grammar, clarity, and concision, as well as generate ad images (Figure 8). We used Claude Code for implementation assistance, including code completion, debugging, and refactoring. The authors reviewed and verified all AI-assisted edit and code for accuracy and correctness.

B Open Science

All artifacts needed to evaluate the core contributions of this paper are publicly available [45], including the code instrumentor, the security service implementations, all study applications with attack integrations, the scripts and data for the effectiveness, performance, and user studies (with the qualitative codebook, survey, and interview protocol), and reproduction instructions.

C Ethical Considerations

In our university IRB-approved study, we pre-screened participants for conditions risking adverse VR effects (e.g., motion sickness, migraines, epilepsy). No recruited participants had such conditions. Participants provided informed consent, were trained on a basic WebXR application [3], and could pause or withdraw anytime without losing compensation. The attacks ran in our controlled environment and involved no personally identifiable information or credentials. To avoid bias, we disclosed the attacks only in a debrief during the interview, after which participants could provide or withdraw consent for their data. Recordings were deleted after analysis; transcripts were anonymized, screening data was never linked to study data, and all data resides on university-authenticated storage accessible only to the research team. Our work analyzes 15 UI-based attacks, including one newly identified, with policies to detect them at runtime, so we believe the benefit outweighs the disclosure risk.