Relay and Betray: Exploiting Client-Side Authority in Multi-User Mixed Reality

Mutahar Ali
University of California, Irvine

Habiba Farrukh
University of California, Irvine

Abstract

Multi-user mixed reality (MR) apps create shared immersive environments where users interact through embodied avatars and virtual objects in real time. These apps are increasingly used in education, workforce training, enterprise collaboration, and social experiences. To maintain immersion, multi-user MR apps minimize latency by making clients compute and broadcast state updates, such as object tracking and interaction updates. Many apps also let users upload user-generated content, such as custom avatars and 3D assets, which is distributed to other participants. This design implicitly assumes client-side trust, where each client is expected to behave honestly: report correct state updates and upload well-formed content. However, a malicious user can exploit this trust to access information that should not be perceptually or logically available to them, inject false updates, bypass moderation and safety features, or upload malicious user-generated content. While prior work has demonstrated side-channels, perception manipulation, and privacy attacks on MR systems, the security implications of client-side trust in shared MR environments remain largely unexplored.

In this paper, we present the first systematic analysis of security risks introduced by client-side trust in multi-user MR apps. We model how users and virtual objects are represented and synchronized across clients, derive security invariants that capture the conditions for safe and correct interaction, and develop a methodology to test for violations via runtime instrumentation and malicious user-generated avatars. Applying this methodology to 20 popular multi-user MR apps across different use cases and platforms, we identify five attack categories: (1) video and audio surveillance, (2) tampering of virtual objects, (3) circumvention of safety features, (4) disruption and denial-of-service, and (5) impersonation of other users. Our findings show that architectural reliance on client-side trust in multi-user MR allows a single malicious user to compromise the privacy, security, and safety of other users.

1 Introduction

Multi-user mixed reality (MR) apps allow users to inhabit shared, persistent virtual environments that blend virtual content with the physical world. Beyond their origins in gaming and entertainment, these immersive apps are increasingly being adopted for education, workplace meetings, social events, and workforce training [1–3, 75]. Unlike traditional teleconferencing, multi-user MR apps provide a heightened sense of co-presence, facilitating real-time interaction through embodied avatars, collective manipulation of shared 3D objects, and communication via spatialized audio.

To deliver a seamless, immersive user experience, multi-user MR apps must meet stringent real-time requirements, continuously synchronizing head, hand, and interaction data at high frequencies (e.g., 60–90 Hz) within tight end-to-end latency budgets (tens of milliseconds) to maintain perceptual stability and avoid motion sickness [4, 65, 98]. To meet these constraints, multi-user MR apps make a critical architectural tradeoff. Instead of relying on authoritative servers that centrally validate, compute, and synchronize all interactions, they adopt client-authoritative or peer-to-relay designs [5, 71]. In this model, each client (user device) performs physics, interaction, and rendering computations locally, then broadcasts its self-determined state updates (e.g., avatar pose, object transforms) to a lightweight relay server, which forwards them to other clients without semantic validation. This design meets the demands of immersion by offloading computation from the server, and with it, trust, to individual users' devices.

However, this performance-driven distributed trust model implicitly assumes that each user behaves honestly. The core logic determining a user's perception of the shared virtual world, i.e., what they see and hear, and how they interact, runs locally, on devices the app cannot control. At the same time, many apps also allow users to introduce custom content, such as avatars and objects, that is automatically distributed and rendered on other participants' devices [6–8]. A malicious participant can therefore modify their client or inject untrusted content to subvert this logic by reading replicated

state that should be perceptually hidden, forging updates that violate interaction constraints, bypassing safety and moderation controls, or destabilizing sessions for other participants. A single attacker can hence compromise the privacy, integrity, and safety of an entire shared MR experience.

Prior research has extensively explored privacy risks in MR, primarily focusing on passive side channels such as inferring sensitive user data from sensor telemetry [59, 81, 82, 87, 88, 95] or network traffic patterns [90, 93]. Other studies have identified implementation vulnerabilities in individual apps [94] or demonstrated client-side attacks by a malicious participant in web-based metaverses [80]. However, these works have overlooked the systemic security implications of client-side trust in popular, native multi-user MR apps. The robustness of these apps against an active, malicious participant who deliberately manipulates their own client to harm others remains largely unexplored.

To address this gap, we present the first systematic security analysis of client-side vulnerabilities in multi-user MR apps. We model how users, avatars, objects, and world instances are represented and synchronized across clients. Guided by app documentation and safety guidelines, we then derive five security invariants that characterize secure and correct shared MR interactions: perceptual secrecy, state integrity, safety enforcement, availability, and identity integrity. Finally, we map each invariant to concrete client-side enforcement logic and test whether a malicious participant can violate it via two attack vectors: (i) runtime manipulation of client logic and state, and (ii) abuse of user-generated avatar pipelines.

We apply this methodology to 20 popular multi-user MR apps across different use cases (e.g., education and training, social interaction, gaming, healthcare, design, and collaboration) and platforms (Quest, Vive, PC VR, Android). We find that all apps exhibit exploitable trust boundaries, allowing a malicious participant to violate core security invariants. By exploiting these boundaries, we demonstrate eight attacks, spanning five main categories: (1) surveillance (20/20 apps), covertly observing or eavesdropping on users beyond intended visual or auditory bounds; (2) tampering (20/20 apps), illegitimately modifying shared virtual objects' state; (3) safety circumvention (15/20 apps), bypassing app-provided safety controls, such as blocking features or host-moderation commands; (4) disruption (12/20 apps), degrading or terminating sessions for other users through resource abuse; and (5) impersonation (6/20 apps), stealing and reusing other users' custom avatars to convincingly assume their identity. Together, these findings expose a fundamental tension in multi-user MR system design: the decentralization that enables immersive, low-latency interaction also undermines the security and safety guarantees of these systems, eroding the trust users place in shared virtual experiences.

In summary, we make the following contributions:

- We conduct the first comprehensive security analysis of client-side trust in multi-user MR apps, exposing how the lack of server-side validation and unvetted user-generated content introduce systemic vulnerabilities that undermine user privacy, security, and safety.
- We formalize a shared state model for multi-user MR apps, derive security invariants defining safe and correct interaction from usage and safety guidelines, and develop a methodology to find and test violations through runtime manipulation and user-generated content.
- We evaluate 20 popular multi-user MR apps across different use cases and platforms, and identify five categories of attacks, including surveillance, tampering, safety circumvention, disruption, and impersonation.

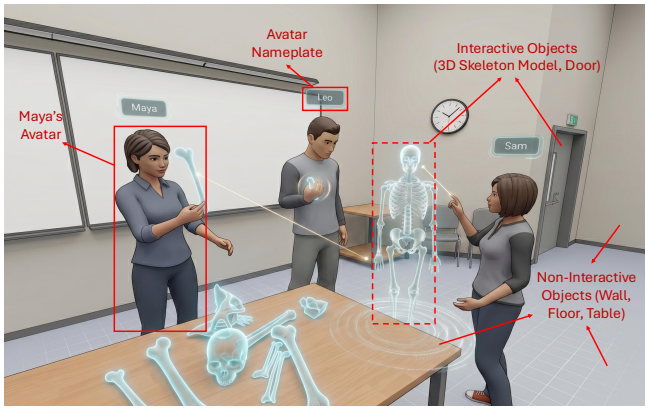


Figure 1: Overview of a multi-user MR world. Users appear as avatars with nameplates and interact with shared objects in a synchronized scene containing both interactive and non-interactive elements.

2 Background

Multi-user MR Apps. Multi-user MR apps enable groups of users to share and experience the same 3D virtual space in real time, typically through head-mounted displays (HMDs) that track their head and hand movements [9, 96]. These apps support a range of use cases, including education and training (e.g., EngageVR, VictoryXR Labs), social interaction (e.g., VRChat, ChilloutVR), gaming (e.g., Gorilla Tag, Rec Room), healthcare (e.g., Nanome, 3D Organon XR), and design and collaboration (e.g., Gravity Sketch, Arkio). For instance, EngageVR is used by corporations such as Bank of America and HSBC, and Kia for onboarding and sales training [10], and by several universities, including for training medical students [11].

These apps host persistent, shared virtual environments called *worlds*, each defining the geometry, lighting, and interactive logic of a virtual space such as a classroom or concert hall. Worlds can be purely virtual or mixed, blending digital objects with live representations of users' physical surround-

ings through passthrough or spatial mapping [12]. Figure 1 summarizes key components of a multi-user MR world, including users, avatars, and objects in a synchronized scene.

To scale participation, apps create instances, which are independent sessions of a world with synchronized state shared among a group of users [13]. Instances may be public or limited to invited participants [13], and each user operates a client on their device that renders the world and synchronizes with others over the network.

Users, Avatars, and Interactions. A user is represented by an avatar in multi-user MR, that mirrors their head, hand, and body movements, conveying identity through appearance and display names [96]. Users interact through embodied actions such as walking, gesturing, grabbing objects, and speaking, with spatialized audio reproducing voice direction and distance cues to reinforce co-presence [14, 15].

Many multi-user MR apps also support user-generated content (UGC), letting users upload fully custom 3D avatars with arbitrary meshes, shaders, animations, and scripts [16]. Such assets are synchronized across clients like any other object.

Objects and Scene. The shared environment within a world is referred to as the scene and includes all virtual objects, both interactive (e.g., avatars, props) and non-interactive (e.g., static scenery, environmental geometry). Interactive objects expose state variables such as position, animation, and ownership that must remain consistent across participants; clients continuously stream their local updates at 60–90 Hz to keep this state temporally and spatially consistent [65, 98]. To minimize latency, this synchronization is typically handled through lightweight relay servers that forward updates without validating their content [5, 71], as we detail in Section 3.1.

Safety Features. To promote safe interaction, multi-user MR apps implement moderation and safety mechanisms such as personal safety bubbles, mute and block options, and in-world reporting tools [62, 103]. Some apps also designate a host with elevated privileges, such as the ability to mute, kick, or restrict participants within a session [17].

3 Problem Statement and Threat Model

3.1 Client-Side Trust in Multi-User MR

To support real-time interaction with minimal latency, which is a key requirement for perceptual stability and immersion [4, 65, 98], multi-user MR apps rely on a lightweight peer-to-server relay networking architecture that forwards state updates between clients without validating or recomputing world state [5, 18, 71]. Unlike an authoritative server model, common in collaborative apps (e.g., Google Docs) and some online games (e.g., Fortnite), where the server validates all client inputs and computes the canonical state centrally [5], the relay model places trust in each client to honestly report its own state. Each client simulates its own local copy of the virtual world, and self-reports updates describing user pose, object motion, and interactions, which the relay disseminates without semantic checks. As a result, a client's self-reported state is taken at face value by all participants. Adding server-side validation on top of this relay is itself impractical: MR's $<$20 ms latency budget cannot accommodate continuous server-side checks on 60–90 Hz pose, voice, and object updates, and the legitimate interaction space, spanning embodied gestures, free-form object manipulation, and user-generated avatars and scripts, is far broader than the narrow input channels validated in traditional online games.

3.2 Threat Model

We consider an adversary who is a legitimate participant in a multi-user MR session and aims to harm other participants. Multi-user MR apps are open systems where any account holder can (a) join public sessions with strangers (e.g., conferences, concerts) or (b) join private sessions as an untrusted invitee (e.g., co-workers, students, contractors, event attendees). This classic insider threat model is widely adopted in prior works exploring the security of other multi-user systems against a malicious user, including videoconferencing [77], online games [64], WebXR [80], multi-user smart home IoT [72], and online platforms more broadly [92].

The adversary's goals are to (1) infer private or occluded information about other users and the environment (e.g., user locations or hidden interactions), (2) manipulate shared world state or ownership to disrupt collaboration (e.g., overwrite objects or desynchronize views), (3) bypass safety or moderation features (e.g., block or kick), (4) degrade availability or terminate sessions for others, and (5) impersonate other users.

The adversary operates a standard client on a commodity MR device with no privileged access to the relay, authentication systems, or other participants' devices. They have two capabilities, both available to every legitimate user: **(A1)** accessing their own client (running on their own device) to inspect memory, modify code, read local files, and capture or send arbitrary but syntactically valid network traffic; and **(A2)** uploading user-generated content (e.g., avatars, worlds, scripts), where supported, through the app's normal creator pipeline. We detail how the adversary exploits these capabilities in Section 5.

We do not consider adversaries that compromise relay infrastructure, break app cryptography, exploit hardware or OS vulnerabilities on other devices, or rely on hardware-based side channels or network-level attacks. We assume the adversary uses a legitimately-created account on the target MR app; we do not consider credential theft.

The adversary targets app-level synchronization and client-side enforcement logic, and may choose any client platform (e.g., Android-based HMDs, PC clients, or mobile devices) supported by the target app. Because the relay protocol is platform-agnostic, an attacker on one platform can affect

victims on any other (e.g., an attacker on Quest can affect victims on Apple Vision Pro or HTC Vive devices).

3.3 Problem Statement

Given that enforcement of interaction rules and safety policies is delegated entirely to clients in multi-user MR, each user's security depends on the honesty of every other client in the shared MR world. We identify four root causes that make multi-user MR apps systematically vulnerable to the adversary described in Section 3.2: the relay's lack of validation, over-replication of sensitive state, unvetted user-generated content, and the delegation of privileges to untrusted clients.

C1. Insufficient Validation and Enforcement. The relay server forwards packets without verifying their authenticity or plausibility, and key safety and moderation rules are enforced locally on each client. The adversary can therefore inject forged updates (e.g., spoofing object ownership or avatar position) or disable enforcement mechanisms (e.g., blocking or muting), causing inconsistencies in world state and policy violations that remain undetected.

C2. Excessive State Replication. To maintain responsiveness, clients receive detailed state information about nearby and occluded users and objects (e.g., behind walls or outside their field of view). This replication enables smooth rendering but exposes private motion, voice, and spatial data that an adversary can inspect, revealing sensitive behavioral or environmental details about other users.

C3. Unvetted User-Generated Content (UGC). Multi-user MR apps support user-created content (e.g., avatars, objects) that is automatically distributed and synchronized across clients. A malicious creator can embed malformed, misleading, or resource-intensive assets that affect the behavior, performance, or perception of other participants.

C4. Instance Master Privilege Abuse. Some apps delegate coordination to a designated client, the *instance master*, which holds elevated privileges (e.g., is trusted for object ownership and moderation events) for managing shared state, even if it is untrusted. These privileges exist only at the client and protocol level, and the user operating the instance-master client is granted no user-facing capabilities beyond those of other participants. A malicious participant can nevertheless deliberately become or impersonate the instance master to exploit the underlying client privileges.

Motivating Example. To illustrate how these root causes translate into concrete threats, consider a professional conference held in a multi-user MR app. The conference session takes place in a shared auditorium world where presenters display slides and attendees gather in breakout groups, and the adversary joins as a legitimate participant. First, exploiting excessive state replication (C2), the adversary inspects replicated pose and audio buffers to infer hidden information, such as attendees' movements in private backstage areas or fragments of side conversations never meant to be shared. Second, exploiting insufficient validation (C1) and instance master privileges (C4), the adversary injects forged updates to overwrite a presenter's slides, seize control of shared tools, or desynchronize participants' views. Third, because safety and moderation controls (e.g., mute, block, or kick) are enforced locally (C1), the adversary bypasses them entirely (e.g., continuing to see other users after being blocked) or spoofs moderation actions. Fourth, the adversary introduces malicious user-generated assets (C3), such as resource-intensive or cloned avatars, to disrupt the session or impersonate users.

Research Gap. Prior security research has focused on passive inference from sensor telemetry [81, 82, 95] or network traffic [90, 93], or on individual app vulnerabilities [94]. A recent work [80] studied privacy and integrity threats from a malicious participant in web-based metaverses, where clients are sandboxed JavaScript running in a browser, and shared virtual state is accessible via browser developer tools. However, the dominant deployment of multi-user MR is native rather than web-based, particularly for high-stakes use cases such as professional collaboration, education, and healthcare.

Native MR differs from the web setting in two key ways: (i) clients are hardened Unity/IL2CPP binaries, with anti-cheat, runtime attestation, certificate pinning, and stripped symbols, rather than JavaScript programs; thus browser-based memory-diffing cannot be used to recover client state; and (ii) native apps expose attack surfaces absent from the web setting (UGC avatar pipelines, instance-master privileges, in-app host moderation). No prior work has systematically examined whether an active adversary can exploit these architectural root causes (**C1–C4**) in widely deployed, native multi-user MR apps, nor what threats arise from the attack surfaces specific to this setting. This paper presents the first such analysis, using an invariant-guided methodology (Section 4) to identify and validate concrete attacks across popular multi-user MR apps.

4 Methodology

To analyze the security risks of client-side trust in multi-user MR, we develop an invariant-guided security analysis methodology, summarized in Figure 2. First, we select representative multi-user MR apps (Section 4.1) ❶. Second, we construct a shared state model capturing how users, avatars, objects, and world instances are represented and synchronized across clients (Section 4.2) ❷. Third, we derive security invariants from app documentation and safety guidelines that define the conditions for secure and correct interaction (Section 4.3) ❸. Finally, we map each invariant to the client-side logic that enforces it and test whether a malicious participant can violate the invariant through runtime manipulation or malicious user-generated content (Section 4.4) ❹. This approach targets architectural weaknesses common across multi-user MR apps, rather than app-specific implementation bugs.

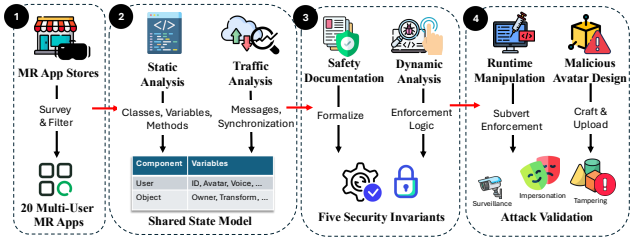


Figure 2: Overview of our methodology. We (1) select 20 representative multi-user MR apps, (2) build a shared state model, (3) derive security invariants from usage and safety guidelines, and (4) test for invariant violations via runtime instrumentation and malicious user-generated content, demonstrating eight attacks across five categories.

4.1 Multi-user MR App Selection

We compiled a list of popular multi-user MR apps by surveying major MR app stores (e.g., Meta Quest, SteamVR, HTC Viveport, PICO), public rankings, news articles, and search results for terms such as "multi-user VR/MR/XR", "social VR/MR/XR", and "collaborative VR/MR/XR". We included apps that are (i) MR-first (i.e., primarily designed for MR rather than flatscreen platforms with optional MR support), (ii) publicly accessible (not enterprise-only or otherwise gated behind organizational access) and available as native apps (not WebXR only) on at least one MR app store, and (iii) multi-user with a social or interactive component, where users share state and interact with one another or with shared objects. For each app, we record supported platforms, primary use cases, popularity indicators, and client development framework.

From this set, we selected 20 apps for in-depth analysis based on popularity metrics (i.e., monthly active users, install counts, user ratings) and maximizing coverage across platforms, development frameworks, and use cases. Appendix A Table 5 lists our complete set of 20 apps. Together, these apps account for 150M+ installs, 280K+ peak concurrent users, and 6M+ monthly active users, multiple platforms (Quest, Vive, PC VR, Android), development frameworks (Unity, native C++), and a diverse range of use cases, including education and training, gaming, social interaction, healthcare, and design and collaboration.

4.2 Modeling Shared State in Multi-User MR

To characterize how shared state is represented and synchronized across MR clients, we analyze network traffic corresponding to in-world interactions and map it to client-side implementation, extracting a unified model of the core components, variables, and relations that define the shared world.

Table 1: Shared state model for multi-user MR. Each component exposes variables synchronized via continuous streams or discrete actions. ↻ = continuous, • = discrete.

Component	Variable	Description	Sync.
User	`userId`	Unique user identifier	•
	`displayName`	Name rendered above avatar	•
	`avatar`	Virtual embodiment of the user	↻
	`camera`	View pose defining the user's FOV	↻
	`voice`	Audio stream from the user	↻
	`audioRange`	Hearing radius	•
	`muted`	Whether the user transmits audio	•
	`blocked`	Blocked user set	•
	`safetyBubble`	Personal-space enabled/disabled	•
Object	`objectId`	Unique identifier for an object	•
	`owners`	Users authorized to control object	•
	`interactive`	Interaction enabled/disabled	•
	`transform`	Position and rotation of object	↻
	`audio`	Object audio stream	↻
	`audioRange`	Audible radius	•
	`appearance`	Mesh/material/shader attributes	•
	`physics`	Physics properties (e.g., velocity)	↻
World	`worldId`	World instance identifier	•
	`access`	Join policy (public/private)	•
	`host`	Host of the world instance	•
	`maxUsers`	Capacity limit	•
	`activeUsers`	Users in the world instance	•
	`objects`	Objects in the world instance	•

4.2.1 State Extraction

We join public and private worlds per app across multiple controlled devices (e.g., Meta Quest 3, Vive Focus 3, PC, Android), drawn from popular and featured listings [19] and selected to span diverse interaction types (e.g., object manipulation, spatial audio, host controls). In each world, we perform canonical actions such as walking, interacting with objects, manipulating menus, and invoking safety controls. We visited up to 20 worlds per app to identify the synchronization protocol and state variables. This was sufficient as we observed that the same message types and client-side data structures recur across worlds, and the set of distinct state variables stabilized well before the final session.

During these sessions, we capture network traffic using Wireshark [20] and AntMonitor [21]. Concurrently, we recover class hierarchies, method signatures, and field names from each app's client binary and hook the relevant methods at runtime to log their invocations, arguments, and return values. Appendix B provides further details on our toolchain, including certificate pinning, symbol recovery, and dynamic instrumentation, used for traffic analysis.

We then analyze the captured traffic to identify message types and fields used for state synchronization, including continuous streams (e.g., avatar pose, voice) and discrete events (e.g., grab, release, block). By cross-referencing message

fields with type and field information recovered from the client binary and with runtime traces, we map network payloads to client-side data structures. For instance, pose update packets contain position and rotation fields originating from the client's avatar tracking state, serialized by the networking layer before transmission. We organize the resulting variables into a unified shared state model, assigning each to the component (`User`, `Object`, or `World`) it describes.

4.2.2 Shared State Model

Table 1 summarizes the shared state model extracted from our analysis. The shared state comprises three components, each maintaining synchronized variables that define the current world instance state.

Core Components. The `User` component represents each participant's identity (`userId`, `displayName`), embodiment and viewpoint (`avatar`, `camera`), and safety settings (`blocked`, `safetyBubble`). The `Object` component represents all manipulable and perceivable entities, including spatial state (`transform`, `physics`), interaction and rendering attributes (`interactive`, `appearance`), and audio properties (`audio`, `audioRange`). The `World` component acts as the coordination context as it maintains membership lists (`activeUsers`), access control and configuration (`access`, `host`, `maxUsers`), and the set of objects defining the scene (`objects`).

Structural Relations. Each user is embodied by an avatar, which is an object with special semantics for identity and perception ($\texttt{User.avatar} \in \texttt{World.objects}$). Each world maintains a set of users and objects ($\texttt{World.activeUsers} \subseteq \texttt{User}$, $\texttt{World.objects} \subseteq \texttt{Object}$), and all synchronized entities belong to exactly one world instance. Object ownership ($\texttt{User.userId} \in \texttt{Object.owners}$) establishes which client is authorized to issue updates for an object's state. Perception is governed by spatial and policy constraints i.e., whether a user can see or hear another depends on `User.camera`, `Object.transform`, occlusion, and `User.audioRange`, all evaluated locally by each client. Safety and moderation are encoded through user relations, such as `User.blocked` and `User.safetyBubble`, which determine which users should be hidden, muted, or kept at a distance.

Synchronization Mechanisms. We identify two synchronization mechanisms for the variables in the shared state: (i) continuous streams for high-frequency data (e.g., pose, audio), broadcast via lightweight UDP-based protocols, and (ii) discrete actions for event-driven updates (e.g., grab, mute, block, host actions), dispatched as remote procedure calls.

4.3 Deriving Security Invariants

To establish targets for our analysis, we derive security invariants that characterize a correct and safe multi-user MR session. Since no formal specification exists for MR apps, we derive these invariants from how these apps are intended to protect users in practice.

Extracting Safety Rules. Prior works have shown that multi-user MR environments are vulnerable to serious security and privacy threats, including harassment, stalking, unwanted proximity, and eavesdropping [58, 61, 69, 85, 103]. In response to these threats, MR apps publish detailed safety and usage documentation, including community guidelines, safety pages, in-app tutorials, host-control documentation, and creator guidelines [22–25], that define prohibited behaviors (e.g., stalking, impersonation) and specify how safety mechanisms should function (e.g., blocked users become invisible and inaudible). We use this documentation as our primary source and verify through our reverse-engineering analysis (Section 4.2) which of these properties are enforced client-side, making them the relevant targets for our analysis.

Mapping Safety Rules to Constraints. For each documented rule or safety feature, we map it to the state variables it governs in our model (Table 1) and express the intended behavior as a formal constraint. For example, spatial audio attenuates voice as a function of inter-avatar distance so that only nearby participants can hear one another [14, 15, 26]; the documented rule governs `User.avatar` and `User.audioRange`, yielding the constraint: for any two users u and v, $v \in \texttt{audibleTo}(u) \Rightarrow \|u.\texttt{avatar.transform.pos} - v.\texttt{avatar.transform.pos}\| \leq u.\texttt{audioRange}$.

Resulting Security Invariants. This process yields five security invariants over shared state variables, each corresponding to a distinct aspect of the shared session that apps explicitly document and enforce: what users can perceive, who can mutate state, whether safety controls are effective, whether session remains stable, and whether identities are protected. Perceptual Secrecy (**I1**) requires that visual and audio data be accessible only when permitted by spatial constraints (e.g., line-of-sight, audio range). State Integrity (**I2**) requires that shared object and world state changes be valid and attributable to authorized owners. Safety Enforcement (**I3**) requires that safety controls (e.g., blocking, host moderation) reliably take effect on every client (e.g., a host's kick removes the targeted user from the session). Availability (**I4**) requires that no client can degrade or terminate a session for others. Identity Integrity (**I5**) requires that a user's visual and nominal identity cannot be cloned or spoofed by another participant. Table 2 specifies each invariant as formal constraints over the shared state model. These constraints are illustrative; concrete implementations may vary across apps (e.g., host controls such as seat locks may be different across apps).

4.4 Identifying Invariant Violations

To understand whether client-side trust can lead to invariant violations, we systematically determine whether each invariant can be violated by a malicious client. We test for violations

Table 2: Security-relevant invariants derived from app usage and safety documentation and expressed as constraints over the shared-state model (Table 1). The listed constraints illustrate representative conditions for each invariant. Concrete implementations and additional controls may vary across apps. u and v denote users while o denotes an object.

#	Invariant	Guidelines	Example Constraints
I1	Perceptual Secrecy	Only see other users or objects when permitted by geometry and viewpoint.	$v \in \texttt{visibleTo}(u) \iff \texttt{inFOV}(u.\texttt{camera}, v.\texttt{avatar}) \land \texttt{LOS}(u.\texttt{camera.pos}, v.\texttt{avatar.pos})$ $\texttt{LOS}(p,q) \iff \neg\exists o \in \texttt{OpaqueObjects} : \texttt{seg}(p,q) \cap \texttt{geom}(o) \neq \emptyset$
		A user should only hear others within a bounded spatial audio range.	$v \in \texttt{audibleTo}(u) \Rightarrow \lVert u.\texttt{avatar.pos} - v.\texttt{avatar.pos} \rVert \leq u.\texttt{audioRange}$
I2	State Integrity	Only authorized users can update shared objects or world state.	$\forall u, o : \texttt{updates}(u,o) \Rightarrow u.\texttt{userId} \in o.\texttt{owners}$
		Users can only take control of objects when permitted, and must relinquish control when a valid transfer occurs.	$u'.\texttt{userId} \in o.\texttt{Owners} \iff \exists u : \texttt{transferOwner}(u,o,u') \land \texttt{allowedTransfer}(u,o,u')$
		Interactions must be proximity- and permission-bounded.	$\texttt{interact}(u,o) \Rightarrow o.\texttt{interactive} \land \lVert u.\texttt{avatar.pos} - o.\texttt{transform.pos} \rVert \leq r$
I3	Safety Enforcement	Blocking and safety features must suppress unwanted interaction.	$v \in u.\texttt{blocked} \Rightarrow v \notin \texttt{visibleTo}(u) \land v \notin \texttt{audibleTo}(u)$ $u.\texttt{safetyBubble} \land \lVert u.\texttt{transform.pos} - v.\texttt{transform.pos} \rVert < r \Rightarrow v \notin \texttt{visibleTo}(u)$
		*Host controls (e.g., locking in seats, summoning users) must be enforced.	$\texttt{seatLock}(u) \Rightarrow \neg\texttt{move}(u)$ $\texttt{summon}(u, pos_s) \Rightarrow u.\texttt{transform.pos} = pos_s$
I4	Availability	A single user should not be able to degrade others' experience or render a session unusable through resource or state abuse.	$\forall u : \texttt{resourceUsage}(u) \leq R_{\max}$ $\forall u : \texttt{updateRate}(u) \leq f_{\max}$ $\forall u, o : \texttt{spawn}(u,o) \Rightarrow \texttt{allowedSpawn}(u,o)$
I5	Identity Integrity	A user's visual and nominal identity should not be easily confused or cloned by another.	$u.\texttt{userId} \neq v.\texttt{userId} \Rightarrow u.\texttt{displayName} \neq v.\texttt{displayName}$ $u.\texttt{userId} \neq v.\texttt{userId} \land \neg \mathit{public}(u.\texttt{avatar}) \Rightarrow u.\texttt{avatar} \neq v.\texttt{avatar}$

*Host controls are app-dependent and may include additional moderation actions beyond the examples shown.

using two main methods: (i) runtime instrumentation of client-side enforcement logic (Section 4.4.1) and (ii) maliciously crafted user-generated avatars (Section 4.4.2).

4.4.1 Runtime Instrumentation

We first map each invariant to the client-side code responsible for enforcing it, then test whether that code can be subverted through runtime manipulation.

Mapping Invariants to Enforcement Logic. For each invariant, we design a minimal in-world scenario that triggers its enforcement (e.g., walking beyond audible range for **I1**, grabbing a shared object for **I2**, blocking a user for **I3**, loading a custom avatar for **I4** and **I5**). While executing these scenarios, we trace method execution and isolate candidate enforcement methods that read or modify the state variables associated with that invariant (Table 2). We then inspect each candidate's decompilation in Ghidra [27] and confirm it as enforcement logic if it performs an explicit check over the relevant state variables (e.g., audio attenuation curves that gate voice output by distance, ownership-transfer callbacks that verify the caller's identity, block-list predicates that suppress rendering of blocked users); methods that only read the variables without acting on them are discarded. This yields, for each invariant and app, a concrete set of enforcement methods. Appendix B provides further details on the toolchain, including symbol recovery and decompilation.

Testing Invariant Violations. We test whether the identified enforcement logic can be subverted using Frida [28] to modify enforcement functions at runtime. For each enforcement method, we select from four manipulation primitives based on the method's role: (i) overriding return values when the method computes a constraint check (e.g., forcing a position query to return a fixed avatar location), (ii) altering function arguments when a parameter governs the constraint (e.g., inflating the audio reception radius), (iii) reading or writing object fields on the calling instance when the enforcement state is stored as an object property (e.g., resetting a seat-lock flag set by a host command), and (iv) replacing function logic entirely when the check is embedded in control flow (e.g., suppressing block-list enforcement via conditional execution). To bypass runtime integrity checks and tampering protections, we strip anti-tamper wrappers (e.g., PairIP) and use a stealth-patched Frida configuration (Appendix B).

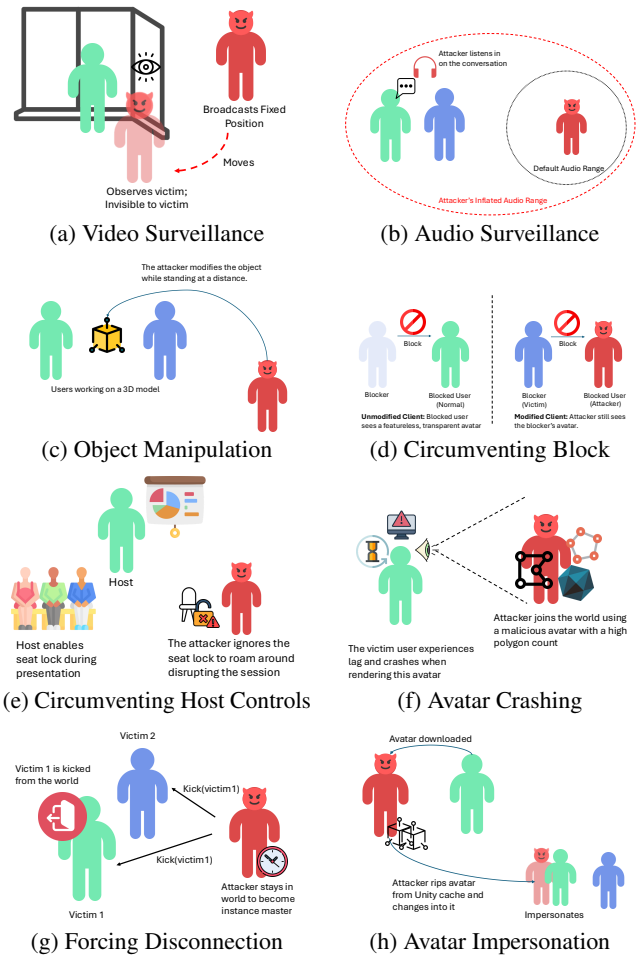
We verify each attack using a multi-device setup. We evaluate our attacks on 4 devices: Meta Quest (the largest market share in standalone MR), HTC Vive (enterprise MR platform), PC VR (high performance on tethered headsets), and Android (foundational platform for standalone headsets), all widely supported by the apps in our study. The attacker device runs a modified client, while the victim and third-party observer devices run an unmodified client in the same world instance. An invariant violation is confirmed when the victim or observer perceives behavior inconsistent with the invariant (e.g., the attacker's avatar appears stationary to the victim while the attacker freely moves and observes). We reproduce each attack across 50 sessions on up to 4 devices, spanning up to 20 different virtual environments (worlds) per app, selected from those featured within the app. We consider an attack successful on an app if it is effective in all 50 trials, confirming that the attack is deterministic.

4.4.2 User-Generated Avatars

User-generated avatars let users design and upload their own embodiment, and often serve as a visual identity marker in MR, similar to a profile picture on a 2D social media platform. This exposes two attack surfaces. First, since avatars are user-designed and rendered on receiving clients, an attacker can craft avatars that elicit adversarial behavior on victim devices. Second, because avatars serve as identity cues, an attacker can impersonate others by replicating their avatars.

Characterizing Avatar Capabilities. We first analyze what each app's avatar system exposes to creators, since this determines which violations are reachable through an avatar. Apps differ substantially in expressiveness. Some apps (e.g., Rec Room, EngageVR, Spatial) restrict avatars to constrained, menu-driven customization with predefined components. Others (e.g., VRChat) expose avatar pipelines that let users upload avatars with arbitrary meshes, shaders, animations, and attached scripts [16, 29]. A few apps (e.g., Vircadia, Overte) accept user-supplied URLs pointing to externally hosted avatar files (e.g., FBX, glTF). For each app, we follow the standard creator workflow and observe how an avatar moves from upload to distribution and how it's cached and rendered on the client. This characterization yields, for each app, the set of avatar capabilities (asset complexity, embedded scripts, custom shaders, etc.).

Testing Avatar-Based Violations. Given a target invariant and an app's exposed avatar capabilities, we test whether a malicious user can use them to violate the invariant via two strategies. First, the attacker designs an avatar to elicit a specific behavior on receiving clients (e.g., resource-heavy rendering, parser failures, or malicious attached scripts), uploads it through the standard creator pipeline, and equips it in a shared world; a violation is confirmed when unmodified victim clients exhibit behavior inconsistent with the targeted invariant. Second, the attacker reuses another user's avatar; since avatars are downloaded to render, every user holds a local copy of those they have rendered. The attacker extracts the target's asset bundle from the cache using public extraction tools [30], re-imports it into a creator project, and re-uploads it under a different account; a violation is confirmed when the clone is indistinguishable from the original to observers.



(a) Video Surveillance (b) Audio Surveillance (c) Object Manipulation (d) Circumventing Block (e) Circumventing Host Controls (f) Avatar Crashing (g) Forcing Disconnection (h) Avatar Impersonation

Figure 3: Visual overview of the attacks.

5 Attacks

After modeling the shared state and deriving security invariants for multi-user MR apps, we apply our methodology for identifying invariant violations (Section 4.4) to the 20 selected multi-user MR apps (Section 4.1), yielding five categories of attacks that enable video and audio surveillance, tampering of virtual objects, circumvention of safety controls, disruption of sessions and interactions, and impersonation of other users. We describe each attack category below, including its underlying vulnerability, how it is realized in practice, and its impact on users across different apps. Table 3 presents the attacks with the invariants they violate, their root causes,

Table 3: Attacks identified by our invariant-guided security analysis. We describe each attack, the root causes (Section 3.3) that enable it, the attacker capabilities (Section 3.2) required to execute it, the security invariants (Table 2) it violates, and prior work i.e., studies of an adjacent attack in a related but different setting (e.g., web-based metaverses, cloud-AR) whose threat model, mechanism, or scope differs from ours. ★ marks new attacks we identify, with no directly related prior work.

Category	Attack	Description	Cause	Cap.	Inv.	Prior Work
Surveillance	Video surveillance	Spoofs broadcast pose to covertly observe other users behind walls or outside FOV.	C2	A1	I1	[79, 80, 95]
	Audio surveillance	Bypasses spatial audio limits to eavesdrop on other users at a distance.	C2	A1	I1	[80]
Tampering	Object manipulation	Forges object-state updates to alter ownership, transforms, or content.	C1	A1	I2	[80, 88]
Safety circumvention	Block evasion	Overrides local block enforcement to keep tracking blocked users.	C1	A1	I3	★
	Host control evasion	Ignores host moderation commands (e.g., kick, seat lock, summon).	C1	A1	I3	★
Disruption & DoS	Avatar crashing	Crashes or stalls nearby clients via a malicious UGC avatar.	C3	A2	I4	★
	Forcing disconnection	Forges moderation messages to disconnect targeted users.	C1, C4	A1	I4	★
Impersonation	Avatar cloning	Extracts another user's avatar from local cache and re-uploads as own.	C3	A2	I5	★

the adversary capabilities they require, and any related prior work in adjacent settings (detailed in Section 7). Figure 3 illustrates each attack, and video demonstrations of all attacks are provided in our artifacts [60].

Table 4 reports which attacks are successful for an app and which platforms the attacker can leverage to execute these attacks, across all 20 apps. For every app except VRChat, we were able to launch attacks on all platforms it supports. VRChat ships Easy Anti-Cheat [31] on its PC VR client and uses attestation on Quest. Nonetheless, we were able to instrument its Vive and Android clients to launch attacks, including against Quest and PC VR users, because the relay is platform agnostic (See Section 6).

5.1 Surveillance

Surveillance attacks allow an adversary to access visual or audio information that should be hidden by spatial or policy constraints, violating the Perceptual Secrecy invariant (I1). These attacks arise because multi-user MR apps broadly replicate the perceptual state to clients (C2) and enforce visibility and audibility locally on the client rather than on the server (C1) for low latency and immersion. Both video and audio surveillance attacks generalize across all apps (Table 4).

5.1.1 Video Surveillance

This attack allows an adversary to forge their broadcast avatar pose, decoupling their true position from the one reported to others, thereby observing other users regardless of walls, distance, or field of view, without revealing their own presence (Figure 3a).

Mechanism and Applicability. Video surveillance succeeds on all 20 apps in our study. Multi-user MR apps continuously stream avatar pose to other clients so that avatars can be rendered immediately as users move or turn. While pose data is broadcast to others, each client computes its own camera position and rendering state. By instrumenting the client, we override the method that emits the outgoing avatar pose so it broadcasts a fixed transform (e.g., a stationary standing position) while the attacker moves normally using standard locomotion controls. The attack generalizes well because many apps build on the same commercial networking engines (e.g., Photon [32]). Similar hooks on Photon's event broadcast (`PhotonNetwork.RaiseEvent`) apply across every Photon-based app (e.g., Spatial, EngageVR). We verify the attack in a multi-device setup, where the other users perceive the attacker as stationary, while the attacker can walk, turn, and reposition their camera arbitrarily to observe nearby users.

Impact. This attack enables stealthy, long-term visual surveillance and stalking in shared MR spaces. The attacker covertly observes participants' actions and body movements while appearing inactive. For example, the attacker can sit silently in a support group in Innerworld, a mental-health peer-support app, and monitor participants who think they're anonymous.

Prior work has shown that captured motion data (head and hand positions) can leak sensitive information, including keystrokes, identity, demographics, and emotional cues [74,

Table 4: Attack applicability across popular multi-user MR apps. ✓ indicates that attacks succeed; ✗ indicates that attacks fail; ❍ means the attack is partially successful; – denotes not applicable (the app lacks the relevant feature, attack surface, or platform availability). We report which attacks are successful for an app (left) and which platforms the attacker can leverage to execute these attacks (right).

	Attacks								Platforms			
App	**Video Surv.**	**Audio Surv.**	**Object Manip.**	**Block Evasion**	**Host Ctrl. Evasion**	**Avatar Crash**	**Forcing Disconn.**	**Imper-sonation**	**Quest**	**Vive**	**Android**	**PC VR**
Gorilla Tag	✓	✓	✓	–	–	–	✗	–	✓	–	✓	✓
Rec Room	✓	✓	✓	✓	–	–	✓	–	✓	–	✓	✓
VRChat	✓	✓	✓	✓	–	✓	✓	✓	✗	✓	✓	✗
Spatial	✓	✓	✓	–	✓	–	✓	–	✓	✓	✓	–
Immersed	✓	✓	✓	✓	–	–	✗	–	✓	✓	–	–
Gravity Sketch	✓	✓	✓	–	–	–	✗	–	✓	✓	–	✓
Cluster	✓	✓	✓	✓	–	✓	✗	✓	✓	–	✓	–
Bigscreen VR	✓	✓	✓	✓	–	–	✗	–	✓	–	–	✓
ChilloutVR	✓	✓	✓	✓	✓	✓	✗	✓	–	–	–	✓
EngageVR	✓	✓	✓	–	✓	–	✗	–	✓	✓	✓	✓
Arkio	✓	✓	✓	–	–	–	–	–	–	✓	–	✓
3D Organon XR	✓	✓	✓	–	✓	–	✓	–	✓	✓	✓	✓
Alcove	✓	✓	✓	–	✓	–	✓	–	✓	–	–	–
Innerworld	✓	✓	✓	–	–	–	✗	–	✓	–	✓	–
ShapesXR	✓	✓	✓	–	–	–	✗	–	✓	–	✓	–
Banter	✓	✓	✓	✓	✓	✓	✗	✓	✓	–	–	✓
Nanome	✓	✓	✓	–	✓	–	✓	–	✓	✓	–	✓
VictoryXR Labs	✓	✓	✓	–	✓	–	✓	–	✓	–	–	–
Overte	✓	✓	✓	❍	✓	✓	✗	✓	–	–	–	✓
Vircadia	✓	✓	✓	❍	✓	✓	✗	✓	–	–	–	✓

81, 82, 90]; our attack strengthens these attacks by placing the attacker's camera directly on the victim through walls and beyond FOV, providing higher-fidelity input for the same inference pipelines without requiring sustained co-location with the victim. Mengascini et al. identified the equivalent attack class in web-based metaverses [80]. We show that the same architectural failure persists in native MR apps, even with stronger hardening.

5.1.2 Audio Surveillance

This attack allows an adversary to eavesdrop on conversations that should be inaudible due to distance or scoping (Figure 3b). Multi-user MR apps typically implement voice communication with spatial audio, where a user's voice is attenuated as a function of distance between avatars, so that only nearby participants can hear one another [14, 15, 26], facilitating small group discussions in large worlds. The attacker's goal is to overhear private discussions beyond the intended spatial audio boundaries without being physically close to the victims.

Mechanism and Applicability. Audio surveillance succeeds on all 20 apps. The server forwards voice streams to users within a broad spatial region, while each client locally computes the attenuation, filtering, and occlusion that determine audibility. By instrumenting the client-side audio pipeline, we bypass these constraints. We find that most Unity-based apps route per-user voice through a Unity audio source with distance rolloff, so overriding its maximum distance parameter (`AudioSource.maxDistance`) defeats falloff regardless of the underlying voice stack (e.g., Photon Voice, Meta PSDK VOIP, WebRTC). The two C++-based apps (Overte, Vircadia) apply attenuation server-side based on the listener's last-reported position; by forging that position, the attacker can still receive the target users' full-volume voice. In each app, the attacker could hear a victim's voice at distances where an unmodified client produced no audio.

Impact. This attack breaks auditory confidentiality in shared MR spaces, where users naturally expect that physical separation limits who can hear them. As a result, users may speak more freely than they would if they knew others could listen. The attack enables passive, undetectable eavesdropping on sensitive conversations. For instance, an attacker can eavesdrop on business meetings in Spatial and Immersed, drug-design discussions in Nanome, and classroom interactions in EngageVR and VictoryXR Labs. The attack can be combined with video surveillance to enable more comprehensive monitoring of both speech and behavior. Mengascini et al. [80] also demonstrated audio surveillance in WebXR (Table 3). We demonstrate the same threat in native MR apps and extend the attack to show that the vulnerability persists even when

filtering is applied server-side.

5.2 Tampering

This attack enables an adversary to tamper with the position, appearance, or interaction behavior of shared objects, violating the State Integrity invariant (I2) (Figure 3c). Objects can be interactive props (e.g., doors, tools, and projectiles) as well as collaborative artifacts (e.g., whiteboards, slides, and 3D models) that anchor shared MR experiences.

Mechanism and Applicability. The attack succeeds on all 20 apps. Across apps, interactive objects are synchronized using a client-authoritative model in which clients are the source of truth for an object's synchronized state, and servers relay updates without enforcing semantic constraints such as proximity or permission (C1). An attacker can exploit this by sending forged state updates and messages. The concrete target varies, but the flaw is uniform. In VRChat and Rec Room, each networked object has a single owner trusted to update its state [33]; ownership transfer is UI-gated by proximity (e.g., by a grab action), but the attacker bypasses the gate by invoking the ownership-transfer call directly and then issues arbitrary updates. Similarly, in Cluster, the messages that grab, create, and destroy interactive objects travel the same broadcast path as pose updates, with no per-message authority check beyond room membership. In Spatial and EngageVR, the attacker dispatches updates outside the intended interaction conditions. For example, writing on a shared whiteboard normally requires standing near it, but the attacker can do so from any distance. In apps with an instance master (VRChat, Rec Room, Spatial), the master is the default owner of all objects [18, 33]; an attacker who intentionally becomes the master (e.g., by outlasting other participants) can retain or reassign ownership across the session, amplifying the impact.

Impact. Object manipulation undermines integrity, safety, and fairness in multi-user MR environments. The impact depends on the use case but is consistently high. In education and training (EngageVR, VictoryXR Labs, 3D Organon XR), an attacker can overwrite a presenter's whiteboard mid-lecture or hide annotations that students rely on. In healthcare and medical education contexts (3D Organon XR, Nanome), the integrity of shared 3D models is critical because researchers and students may rely on the positions of atoms, bonds, and anatomy markers for clinical decisions or assessment (e.g., an attacker can move anatomy pins on a shared cadaver model to teach incorrect anatomy). In design and collaboration, tampering enables targeted misinformation that the victim is unlikely to detect. Mengascini et al. demonstrated object tampering in web-based metaverses [80], and Slocum et al. studied shared-state manipulation in cloud-AR localization via sensor-input spoofing to a trusted server [88] (Table 3). Our attack generalizes beyond these settings: it spans arbitrary ownership transfers and state or content updates, and applies across native MR apps regardless of client framework (Unity or C++).

5.3 Safety Circumvention

Multi-user MR apps expose safety controls, such as per-user blocking and host moderation tools, to help participants avoid harassment and keep sessions orderly [34, 35]. Because many safety features are enforced locally (C1), a malicious client can ignore safety-critical commands or disable local enforcement logic, violating the Safety Enforcement (I3) invariant.

5.3.1 Block Evasion

Many apps provide a block feature to mitigate harassment [62]. When user *A* blocks user *B*, *A* and *B* become mutually invisible (e.g., VRChat) or visually degraded (e.g., Rec Room), and inaudible [17, 34, 35]. This enables the victim (*A*) to remain in the same session while removing the harasser (*B*) from their sensory experience. In this attack, an adversary (*B*) modifies their client to bypass these filters, enabling them to continue seeing and tracking user *A*. Figure 3d shows how the attack works.

Mechanism and Applicability. Block evasion succeeds fully for 7 apps (VRChat, Rec Room, ChilloutVR, Cluster, Immersed, Bigscreen, Banter), and partially for 2 apps (Vircadia, Overte); the rest have no block feature, so the attack does not apply to them (Table 4). The block is enforced locally on each client (C1) by hiding and muting the blocked user for the blocker and vice versa. Yet, apps must still replicate both users' state updates (e.g., movement, object interactions) to all clients (C2) to keep the world consistent. We instrument the app to ignore the block predicate or clear the local block list to continue rendering and tracking *A*'s avatar, while *A* believes they are hidden. For example, in ChilloutVR, the block list is stored in a JSON file on the blocked user's disk and enforced by `ModerationManager.IsBlocked` on each frame, which can be ignored by an attacker.

In Overte and Vircadia, the block list is stored on the server and used to implement a bidirectional filter for avatar pose and voice. However, the server still propagates the blocked user's object edits, chat messages, and friend requests, so a blocked user can still interact with the blocker.

Impact. Harassment is a serious concern in multi-user MR and can be more distressing than on traditional 2D platforms due to the immersive nature of MR [62, 86]. Victims may rely on blocking to create distance or regain a sense of safety [58, 103]. Block evasion undermines this primary safety control, enabling persistent stalking and targeted monitoring without the victim's knowledge. The harm is particularly acute in apps frequented by vulnerable populations, such as seniors (Alcove) and mental health patients (Innerworld), and in workplace settings (e.g., Immersed, EngageVR).

5.3.2 Host Control Evasion

Host-moderated MR apps, designed for education and professional collaboration, such as EngageVR and Spatial, provide

the host with special controls to keep meetings orderly [17]. These include features, such as summoning participants to a specific location of the world, locking them in seats (e.g., during presentations or talks), and removing troublemakers. The attacker's goal is to bypass these moderation commands. Figure 3e illustrates the attack.

Mechanism and Applicability. This attack works on all 10 apps in our analysis that provide instance-level host moderation tools (Table 4). Host commands are delivered as Remote Procedure Calls (RPCs) to each client. The server forwards these commands but does not verify that clients obey them; enforcement occurs on the client (C1), which interprets the RPC and updates locomotion, presenter, or pose-related state accordingly. For example, in EngageVR, a seat-lock command arrives as an RPC (`SitTriggerSetLockedInSeat.ExecuteNode`) that sets a boolean flag (`set_lockedInSeat`) on the client. A malicious client can drop the RPC or write to the corresponding object field afterward, to continue moving freely. The pattern recurs across apps. For example, in Spatial, the kick command from the host to remove a user from the world arrives as an RPC (`RPC_KickUserFromRoom`) and can be ignored by the attacker. We verified the attack using a multi-device setup with the attacker, host, and a third-party observer.

Impact. Evading host session controls disrupts educational, professional, and clinical sessions: an unruly participant can roam during a lecture, ignore relocation requests, or repeatedly disrupt group activities, reducing the host's ability to maintain order. Further, the host is unable to remove this participant from the environment. This is especially damaging in apps such as EngageVR, which targets classrooms and conferences [36], and Spatial, which markets use cases including business meetings, training/onboarding, and brand promotion events [37, 38], and therefore rely on host controls for smooth sessions. In 3D Organon XR, the same primitive lets an attendee disrupt anatomy reviews in medical training.

5.4 Disruption and Denial of Service

A malicious participant can also degrade or terminate other users' ability to participate in a session, violating the Availability (I4) security invariant. By spoofing moderation events, sending malformed updates, or forcing others to render computation-intensive content, they can stall, disconnect, or crash nearby users. These attacks arise because clients apply networked updates, including moderation events, with insufficient validation (C1), apps distribute user-generated content such as custom avatars (C3), and some apps give moderation privileges to an untrusted instance master client (C4).

5.4.1 Avatar Crashing

Custom avatars are a highly desired feature in multi-user MR [102]. However, apps that let users upload fully custom avatars (including arbitrary meshes, shaders, animation rigs, and asset bundles), built in 3D creation software such as Unity or Blender, are vulnerable to an attack that forces nearby users' clients to freeze or crash by loading a malicious avatar deliberately designed to consume excessive resources (Figure 3f).

Mechanism and Applicability. We successfully validated this attack on every app in our study that exposes a fully custom-avatar pipeline (6 out of 20 apps). Avatar assets are distributed to nearby clients for local rendering (**C3**), so a malicious user can craft an avatar to deliberately overload other clients' rendering and physics engines. For example, we implement this attack on VRChat by creating malicious avatars that (1) exhaust hardware resources through expensive shaders, high-polygon meshes, oversized textures, and excessive physics components, (2) trigger engine-level parsing failures via malformed asset bundles, and (3) abuse Udon scripts [29] for network flooding (e.g., sending thousands of sync events) and recursive rendering loops.

To quantify the impact, we designed 10 intentionally poorly optimized avatars and measured their effect on a Meta Quest 3 device in a VRChat session. Loading these avatars caused an average frame-rate drop of approximately 70% on average, sufficient to render the victim's session unusable. Because these failures occur on the victim's device, the attacker can effectively "crash out" nearby users while remaining in the same world. Other apps fail the same way. For example, Cluster accepts standard Virtual Reality Model (VRM) file uploads through its web portal; receiving clients parse these and crash due to malformed blendshapes, recursive node hierarchies, or oversized index buffers. In Overte and Vircadia, the attacker points the receiver's parser at any URL by setting `MyAvatar.skeletonModelURL`.

Notably, apps sometimes enforce format and size constraints and perform checks on the avatars at upload time. For example, VRChat enforces a size limit of 200 MB download size and assigns performance ranks based on static analysis [39]. However, our crasher avatars passed these checks.

Impact. Avatar-crashing enables denial-of-service against shared MR sessions. A single malicious attendee can repeatedly join and crash or severely degrade the experience for nearby users, disrupting classes, enterprise meetings, or live events, and eroding trust in the app. VRChat and Cluster are actively used for live events, including ticketed events (e.g., concerts) [40], where even brief freezes can disrupt large audiences and derail time-sensitive programs. Since the attack uses the same avatar distribution mechanism as normal content and can be executed simply by equipping an avatar, it is easy to repeat, share with others, and combine with other disruptive behaviors (e.g., impersonation or harassment).

5.4.2 Forcing Disconnection

This attack allows an adversary to forge moderation events, such as "kick" messages, to eject other users from a session. Figure 3g depicts the attack.

Mechanism and Applicability. This attack succeeds on 7 apps: VRChat, Spatial, Rec Room, 3D Organon XR, Alcove, Nanome, and VictoryXR Labs (Table 4). We observe two methods to launch the attack. (1) In apps with explicit application-layer kick RPCs (Spatial, VictoryXR Labs, Nanome), the attacker can directly invoke them from their client to spoof kick commands. For example, the `Send_KickUserFromRoom` RPC in Spatial is callable from any client. (2) Where moderation actions are delegated to a designated instance master client (C4) (VRChat, Rec Room), the attacker abuses the master client election process. For example, in Photon PUN [32], a popular, commercial networking engine, when the current master client disconnects, the longest-connected remaining client is automatically promoted. An attacker can exploit this design by manipulating join/leave timing to become the master client and then sending disconnect messages (`PhotonNetwork.CloseConnection`). As a result, the targeted user is removed from the instance. The attack fails for apps where an attacker cannot acquire instance master privileges due to a different instance master selection process. For example, in EngageVR, the instance master is always the host who started the session, and the session ends when the host exits.

Impact. This is a targeted denial-of-service attack, allowing an adversary to eject specific, legitimate participants from social gatherings, educational sessions, or enterprise meetings. In VictoryXR Labs and 3D Organon XR, an attacker can quietly remove students from a class or lab session; Alcove is aimed at older adults reconnecting with distant family, for whom an abrupt, unexplained disconnection is both distressing and hard to diagnose or recover from.

5.5 Impersonation

This attack allows an adversary to clone another user's custom avatar and reuse it, violating the Identity Integrity invariant (I5). Because MR apps distribute avatar assets to clients for rendering (C2, C3), a malicious user can extract and replicate another user's appearance, then join a world with the copied avatar, optionally under a confusingly similar display name (e.g., with a trailing space or underscore). To other participants, the attacker and victim appear nearly indistinguishable, as shown in Figure 3h.

Mechanism and Applicability. We successfully validated this attack on all 6 apps that ship a fully customizable avatar pipeline. When a victim joins a world, their avatar's model, textures, and materials are downloaded to every other client and cached for rendering. An attacker can join the same world as the victim, wait for their avatar to load, and then use publicly available asset ripping tools [30] (See Section 4.4.2) to extract the asset bundle from their local cache.

We verified the attack using three devices: the victim (with the target avatar), the attacker, and an observer. The attacker extracts the victim's avatar from cache and re-uploads it (e.g., to VRChat via standard upload workflow [16]) as a new avatar under their own account. Since the clone is indistinguishable from the original, the observer sees two visually identical avatars with similar nameplates and must rely on subtle cues (e.g., user IDs or voice) to tell them apart. We do not evaluate the attack on apps that constrain avatars to closed catalogs (e.g., Rec Room, EngageVR, and Spatial), since many users share similar avatars due to restricted options.

Impact. Avatar cloning enables convincing visual impersonation in public and semi-public MR spaces, allowing an attacker to pose as a teacher, host, or colleague to mislead users, issue deceptive instructions, or damage the victim's reputation. Because avatars are a primary identity cue in MR, such impersonation undermines trust in social and professional interactions, especially in settings where participants do not know each other personally [76]. For example, in Spatial's brand-promotion events [38], an attacker can impersonate a presenter to misdirect attendees. It also undermines creator rights, since unique avatars are intellectual property that can be copied and redistributed without consent.

6 Discussion

The Immersion-Security Tradeoff. Our analysis demonstrates that all 20 apps are vulnerable to the same classes of attacks, despite differing in their use cases, implementations, target audiences, and feature sets. This is because the vulnerabilities stem from shared architectural choices (C1–C4), not from app-specific bugs. These attacks highlight the fundamental tension between the low-latency, client-driven synchronization that enables immersive MR interaction and the risk that a malicious participant can forge state, eavesdrop, and bypass safety controls. However, our findings also suggest this tradeoff is not all-or-nothing. Not all state updates are equally latency-sensitive. For instance, continuous streams (e.g., head/hand pose) demand client-driven forwarding, but discrete, high-impact actions (e.g., ownership transfers, moderation commands, kick events) can tolerate server-side validation without harming immersion. For example, Overte and Vircadia attempt to enforce blocklists at the server, showing that selective server authority is possible.

Attack Feasibility and Detectability. Our attacks require reverse-engineering the MR app binary and runtime instrumentation via Frida. While this demands moderate technical skill, once an attack script is developed, it can be packaged and distributed, lowering the barrier for subsequent use. The effort varies across apps: a few apps (notably VRChat) adopt substantial runtime anti-tampering protections while most

ship with little or none. Such protections raise the cost of building an attack but cannot prevent it: the attacker fully controls the device the client runs on and can ultimately hook or patch around any client-side check. UGC-based attacks (e.g., avatar crashing) require no reverse engineering at all, only the ability to upload an avatar through the app's standard pipeline. During our testing, none of our test accounts were flagged, warned, or banned by any app, consistent with the architectural expectation that the relay performs no semantic validation and thus does not detect these violations.

An attacker can also use alternative instrumentation tool chains (e.g., LSPosed [41], LLDB scripts [42]), binary patching, or a MITM proxy [43] to launch attacks. We chose Frida because it generalizes across platforms (Android-based HMDs, PC VR) with the same hook definitions, unlike binary patching, which must be repeated per platform, and avoids the latency overhead of a proxy-based interception path. Since our threat model assumes an attacker can generate any syntactically valid protocol message, the attacker can also reimplement the relay protocol as a standalone client.

Attack Generalizability. Our attacks are not dependent on or specific to client development frameworks (e.g., Unity), runtime standards (e.g., OpenXR), or headset refresh rate since they operate at the application layer. We tested them on C++-based apps (Overte, Vircadia), on a device without OpenXR support (Android mobile device), and at different refresh rates (72/90/120 Hz) on a Meta Quest 3, and found no differences in effectiveness.

Why MR Differs from Traditional Domains. The security risks we identify in multi-user MR differ from those in traditional online games or videoconferencing in three ways. First, MR exposes richer sensing and perception pipelines, including 6 DoF tracking, spatial audio, and embodied avatars, creating new attack surfaces absent in screen-based systems. Second, MR's stricter latency requirements ($<$ 20 ms for perceptual stability [4, 65, 98]) further constrain server-side validation. Third, multi-user MR is increasingly deployed in high-stakes settings such as classrooms, enterprise meetings, workforce training, and live events [1–3, 75], where these attacks cause privacy violations, harassment, and sabotage of professional or educational activities rather than mere unfair play.

Defenses. We organize defenses by the root causes they address and provide concrete recommendations.

Selective server-side validation (C1, C4). Adding meaningful server-side validation to a relay-based design is hard. Full validation of continuous pose and audio streams is impractical given MR's latency budget. Even where the checks are affordable, the state space of possible MR updates is huge and legitimate user behavior is hard to specify, unlike in games. However, discrete high-impact actions, such as moderation commands, are infrequent enough to be validated server-side with acceptable overhead. We recommend that apps identify such actions and route them through server approval rather than relying solely on client-side or instance-master authority. Designing such selective, low-latency validation for all MR interactions remains an open challenge.

State minimization (C2). Surveillance attacks exploit the relay's broadcast of the full state to all clients. Server-side spatial filtering i.e., forwarding only the state that should be perceptually available to each client based on position, occlusion, and audio zones, would directly mitigate these attacks, albeit at some cost to performance.

Content pipeline hardening (C3). UGC pipelines should be treated as untrusted input channels with server-side validation, including strict limits on asset complexity (e.g., polygon count, texture size, shader cost), static analysis of embedded scripts, and sandboxed execution [44]. Avatar distribution should use server-side access controls rather than relying on client-side caching, which enables cloning. Partial measures already exist, but leave gaps. For example, VRChat assigns each avatar a performance rank [39] but does not enforce a hard complexity bound, so a client can still load deliberately expensive avatars, and their recent encryption of the on-disk avatar cache on PC does not prevent cloning, since the client must still decrypt the assets to render them, leaving them recoverable on disk. Emerging anti-cloning techniques [44] tend to introduce performance or usability tradeoffs.

Client integrity and attestation (defense in depth). Our cross-platform evaluation (Table 4) shows that the attacks reproduce on every supported platform for all but one app, and on at least one platform for every app. As the relay protocol is platform-agnostic, an attacker who compromises a client on one platform can affect victims using any other supported platform without any extra effort.

Traditional integrity checks (e.g., executable hashing, anti-debugging) and anti-cheat engines [31, 45, 64, 101] raise the bar for attackers but are insufficient as primary defenses in MR. While some MR apps adopt PC-style anti-cheat protections, these are limited to desktop clients; standalone HMDs typically lack kernel-level enforcement, and such mechanisms raise additional privacy and performance concerns. Basic checks can be bypassed through binary patching or runtime instrumentation. Remote attestation [46, 84] is similarly limited: it is often bypassed on rooted devices [67], misconfiguration is common [73], and many commodity HMDs do not expose hardware-rooted attestation to third-party developers. Moreover, attestation is typically performed at app startup and cannot detect runtime modifications such as injected hooks. For example, VRChat ships Easy Anti-Cheat on its PC VR client and uses remote attestation on Quest [47]; however, the same protections are not available on the Vive headset or an Android mobile device (Table 4). The attacker can simply instrument the Vive or Android mobile client to launch the attacks, including against PC VR and Quest users, due to cross-platform lobbies. These mechanisms are best deployed as defense in depth alongside the server-side measures above.

Limitations. Our evaluation covers 20 multi-user MR apps spanning different use cases, platforms, and popularity tiers. While 18 of the 20 apps are built on Unity, reflecting its dominance in multi-user MR development [48], we also evaluate two open-source C++-based apps; our attacks also succeed on them, indicating that the vulnerabilities arise from architectural assumptions common to relay-based multi-user MR rather than from framework-specific bugs. Extending our analysis to other engines (e.g., Unreal [49]) is future work. We assume that the adversary can join or remain in a session, either publicly or via invitation. Some high-stakes settings, such as closed enterprise deployments and first-responder training [3], may impose stricter authentication and access controls, but insider threats and compromised devices remain realistic there. Our attacks were tested on specific app versions; apps may patch individual issues, but the root causes are architectural and persist across versions. Finally, our invariants reflect current MR features; as apps introduce richer sensing or new safety abstractions, additional invariants and attack classes may emerge, which our framework can accommodate by mapping them to the corresponding client-side logic enforcing them.

7 Related Work

Security and Privacy in MR. Prior work has extensively explored single-user, passive inference threats, including side-channel attacks. Studies have shown that client telemetry can leak sensitive information. Head and hand movements enable user identification [81] and personal attribute inference [82], aggregated sensor streams allow near-perfect fingerprinting [74], and additional leakage including keystrokes, blood pressure, and user activities, occurs through head motion [59, 87], spatial maps [66], motion and position sensors [97, 99], controller acoustic emanation [78], and GPU profiling [89]. These works assume a passive adversary that is either co-located or has access to raw sensor data. In contrast, our adversary is a legitimate participant in the shared world with no access to sensor data or the victim devices, who exploits the lack of server-side validation in native MR apps.

A separate line of work analyzes MR apps through static or network analysis, uncovering excessive permission requests [68], privacy policy inconsistencies, and sensitive data exfiltration [93, 100]. These works are focused on analyzing potentially malicious app behavior, whereas we explore vulnerabilities arising from modified app clients.

Security and Privacy Threats in Multi-user MR. Recent work has begun to explore security in multi-user MR settings. Vondracek et al. demonstrated attacks exploiting application-specific flaws to join private virtual rooms [94]. Other studies have demonstrated sensitive inference in multi-user environments, including de-anonymization of avatars using gait [79] and keystroke inference from unencrypted motion traffic [90] and avatar views [95]. These works exploit specific implementation bugs [94] or perform passive inferences from unencrypted motion traffic and avatar views [79, 90, 95]. In contrast, our work targets architectural vulnerabilities common to native multi-user MR rather than app-specific bugs and demonstrates active manipulation of shared state.

Another work studied shared-state manipulation attacks in cloud-AR localization systems (e.g., ARCore Cloud Anchors, Google VPS), where a server resolves hologram positions from user-supplied GPS/camera/IMU data. The attacker spoofs these inputs to misplace or exfiltrate holograms [88]. Contrarily, our work explores real-time multi-user MR sessions rather than cloud-AR localization, and exploits shared-state synchronization between clients rather than sensor inputs to a trusted server.

A recent work explored privacy and integrity threats from a malicious participant in web-based metaverses using JavaScript memory diffing via browser developer tools [80]. However, the native MR setting differs from web metaverses in client substrate, hardening profile, and deployment stakes, and our work extends their analysis along three axes: attacks, methodology, and scope. First, we demonstrate five new attacks and exploit attack surfaces specific to native MR (e.g., UGC avatar pipeline, instance master privileges). Second, their methodology does not apply to native apps: (i) native MR apps ship integrity protections absent in web-based metaverses (RASP, remote attestation, signature verification, anti-cheat, certificate pinning) that must be bypassed before analysis, and (ii) native apps do not expose a global window object; memory is isolated across processes, symbols are stripped, and diffing OS-level snapshots does not converge in practice due to app scale and dynamic allocation. Instead, we reverse-engineer apps and instrument enforcement logic directly; our approach applies uniformly across platforms and worlds regardless of world content or size, enabling reliable and repeatable attack execution. Third, our evaluation spans 20 native apps across various use-case categories, including enterprise, healthcare, and education deployments, where these threats represent serious harm. We provide the first systematic analysis linking the client–relay architecture of native multi-user MR apps to both read-based and write-based violations.

Security and Privacy Threats in Traditional Multi-user Applications. Client-side manipulation has also been studied in traditional multi-user systems, including "Zoombombing" in videoconferencing [77] and cheating in online games [63, 64, 83, 91, 101], where attackers use aimbots or wallhacks for competitive advantage [64, 70] and defenses focus on detecting such manipulations [83, 91, 101]. These efforts primarily target fairness violations over narrow input/output channels such as keyboard and mouse. Multi-user MR apps face stricter latency requirements and expose richer sensing pipelines, real-time spatial interactions, and immersive social contexts, creating distinct classes of privacy, integrity, and safety risks.

8 Conclusion

Multi-user mixed reality (MR) apps are increasingly used for social, educational, and enterprise interaction. Yet their architectures still implicitly trust client devices. In this work, we present a systematic security analysis of this client-side trust. We model the shared state of multi-user MR apps in terms of users, objects, and worlds, and derive five security invariants capturing perceptual secrecy, state integrity, safety enforcement, availability, and identity integrity. We systematically test whether these invariants can be violated by a malicious participant through runtime manipulation of client-side logic and abuse of user-generated avatar pipelines. We evaluate on 20 representative MR apps, across different use cases and platforms, to demonstrate five classes of client-side attacks, namely surveillance, tampering, circumvention of safety features, disruption, and impersonation, that allow a single malicious participant to compromise others' security, privacy, and safety. Our shared-state model and invariant-guided methodology provide a general framework for analyzing future MR features and apps, and our findings highlight the risks of treating MR clients as trusted and motivate defenses that combine selective server authority, context-aware validation, and hardened user-generated content pipelines to improve the security of multi-user MR environments.

Acknowledgments

We thank our reviewers and shepherd for providing us with valuable feedback. This work has been partially supported by the UCI Information and Computer Science (ICS) Research Award. Any findings, conclusions, and recommendations expressed in this paper are those of the authors only.

Ethical Considerations

Stakeholders. We identify four key stakeholder groups potentially impacted by this research:

End users. End users are the primary beneficiaries of our work. Our attacks expose concrete ways adversaries can harm users in shared MR sessions, including surveillance, harassment, impersonation, and disruption. We provide recommendations to help platforms design effective defenses.

App Developers. We have initiated responsible disclosure to the developers of the apps in our analysis and will complete it with sufficient lead time before publication.

Content creators. Avatar cloning undermines creator rights by enabling unauthorized extraction and use of original work. We do not extract or redistribute any real creator's avatar, and our findings would motivate platform-level protections.

Research community. The shared-state model and invariant-guided methodology are reusable for future work on security modeling of new multi-user MR features and apps.

Study Ethics and Mitigations. All experiments were conducted on devices and accounts we owned, in private sessions, without interaction with unsuspecting users or the generation of load that could affect platform infrastructure. We reviewed each platform's Terms of Service and limited our activity to the minimum necessary for the research. No real-user data was collected. Only activity from our own test accounts was logged. To prevent operational misuse, we provide examples of class, method, and variable names sufficient to convey our methodology, while omitting app-specific offsets, symbol maps, and ready-to-run exploit scripts.

Justification for Research and Publication. The vulnerabilities we identify stem from architectural assumptions common to relay-based multi-user MR apps, which a similar analysis could independently surface. Disclosing and analyzing these issues within the research community, in coordination with affected vendors, enables developers to address them proactively before they can be widely exploited. We judge that the long-term benefit to end users from motivating architectural defenses outweighs the incremental attacker uplift, given coordinated disclosure and our restraint on operational detail. We hope our work contributes to the design of secure and trustworthy multi-user MR platforms, as these systems are increasingly adopted for high-stakes use cases.

Open Science

To support reproducibility, we publish all the artifacts for our reverse engineering and instrumentation pipeline, including scripts for binary analysis, logging framework, and attack execution [60]. To prevent operational misuse, we withhold platform-specific exploit code from public release, since fully operational exploit scripts would significantly lower the barrier for malicious use and, released before fixes are deployed, would be inconsistent with responsible disclosure. We have disclosed all identified issues to the affected vendors and are following coordinated disclosure practices. We will make the complete set of artifacts, including platform-specific exploit code, available to other researchers upon request.

References

[1] EngageVR: Virtual Reality Education Services. https://engagevr.io/. [Accessed: 25-May-2026].

[2] Can AR and VR finally disrupt the exhausting culture of video meetings? https://www.bbc.com/worklife/article/20240125-can-ar-and-vr-finally-disrupt-the-exhausting-culture-of-video-meetings. [Accessed: 25-May-2026].

[3] Virtual Reality (VR) Training Systems for First Responders Market Survey Report. https://www.dhs.gov/sites/default/files/2024-07/2024_0709_st_vrmsr%20%282%29.pdf.[Accessed: 25-May-2026].

[4] What networking challenges are unique to VR multiplayer applications? https://milvus.io/ai-quick-reference/what-networking-challenges-are-unique-to-vr-multiplayer-applications. [Accessed: 25-May-2026].

[5] Authoritative Servers, Relays & Peer-To-Peer: Understanding Networking Types and their Benefits for Each Game Type. https://edgegap.com/blog/explainer-series-authoritative-servers-relays-peer-to-peer-understanding-networking-types-and-their-benefits-for-each-game-types. [Accessed: 25-May-2026].

[6] Creating Your First World in VRChat. https://creators.vrchat.com/worlds/creating-your-first-world/. [Accessed: 25-May-2026].

[7] Spatial Creator Toolkit. https://toolkit.spatial.io/. [Accessed: 09-Nov-2025].

[8] Horizon Worlds desktop editor. https://developers.meta.com/horizon-worlds/learn/documentation/welcome-creators. [Accessed: 25-May-2026].

[9] Shared experiences in mixed reality. https://learn.microsoft.com/en-us/windows/mixed-reality/design/shared-experiences-in-mixed-reality. [Accessed: 25-May-2026].

[10] Virtual and Augmented Reality for Team Building and Onboarding. https://engagevr.io/virtual-and-augmented-reality-for-team-building-and-onboarding/. [Accessed: 24-May-2026].

[11] University of Miami – ENGAGE Case Study. https://engagevr.io/university-of-miami-engage-case-study/. [Accessed: 24-May-2026].

[12] Meta Passthrough API Overview (Unity). https://developers.meta.com/horizon/documentation/unity/unity-passthrough/. [Accessed: 25-May-2026].

[13] VRChat Instances. https://wiki.vrchat.com/wiki/Instances. [Accessed: 25-May-2026].

[14] VRChat Spatial Audio Source. http://creators.vrchat.com/worlds/components/vrc_spatialaudiosource/. [Accessed: 25-May-2026].

[15] Rec Room Directional Voice Chat. https://blog.recroom.com/posts/2020/10/26/directional-voice-chat. [Accessed: 25-May-2026].

[16] Creating Your First VRChat Avatar. https://creators.vrchat.com/avatars/creating-your-first-avatar/. [Accessed: 25-May-2026].

[17] ENGAGE VR Host Controls. https://docs.engagevr.io/engage/host-controls. [Accessed: 25-May-2026].

[18] Spatial Multiplayer. https://toolkit.spatial.io/docs/multiplayer/overview. [Accessed: 25-May-2026].

[19] VRChat Worlds. https://vrchat.com/home/worlds. [Accessed: 25-May-2026].

[20] Wireshark. https://www.wireshark.org/. [Accessed: 25-May-2026].

[21] AntMonitor. https://athinagroup.eng.uci.edu/projects/antmonitor/. [Accessed: 25-May-2026].

[22] VRChat Community Guidelines. https://hello.vrchat.com/community-guidelines. [Accessed: 25-May-2026].

[23] Rec Room Code of Conduct. https://recroom.com/code-of-conduct. [Accessed: 25-May-2026].

[24] Your Safety – ENGAGE VR. https://docs.engagevr.io/engage/getting-started/your-safety. [Accessed: 25-May-2026].

[25] Spatial Community Guidelines. https://www.spatial.io/guidelines. [Accessed: 25-May-2026].

[26] ENGAGE VR Documentation. https://docs.engagevr.io/engage. [Accessed: 25-May-2026].

[27] Ghidra: Software Reverse Engineering Framework. https://github.com/NationalSecurityAgency/ghidra [Accessed: 25-May-2026].

[28] Frida – A dynamic instrumentation toolkit. https://github.com/frida/frida [Accessed: 25-May-2026].

[29] VRChat Creator Documentation. https://creators.vrchat.com/worlds/udon/. [Accessed: 25-May-2026].

[30] AssetRipper: Open-source tool to extract and explore Unity assets. https://github.com/AssetRipper/AssetRipper. [Accessed: 25-May-2026].

[31] Easy Anti-Cheat. https://www.easy.ac/en-US. [Accessed: 25-May-2026].

[32] Photon Unity Networking (PUN). https://www.photonengine.com/pun. [Accessed: 25-May-2026].

[33] VRChat Networking: Object Ownership. https://creators.vrchat.com/worlds/udon/networking/ownership/. [Accessed: 09-Nov-2025].

[34] Blocking, Unblocking, and Reporting Users in VRChat. https://help.vrchat.com/hc/en-us/articles/43880369459603-Blocking-Unblocking-and-Reporting-Users-in-VRChat. [Accessed: 25-May-2026].

[35] Blocking Other Players in Rec Room. https://recroom.zendesk.com/hc/en-us/articles/4533138815511-Blocking-Other-Players. [Accessed: 25-May-2026].

[36] ENGAGE VR Virtual and Augmented Reality Use Cases. https://engagevr.io/virtual-and-augmented-reality-use-cases/. [Accessed: 25-May-2026].

[37] Four Uses of Virtual Reality in Spatial. https://www.spatial.io/blog/four-uses-of-virtual-reality-in-spatial. [Accessed: 25-May-2026].

[38] Brands – Spatial. https://www.spatial.io/brands. [Accessed: 25-May-2026].

[39] VRChat Avatar Performance Ranking System. https://creators.vrchat.com/avatars/avatar-performance-ranking-system/ [Accessed: 11-June-2026].

[40] Exploring the VR Music Revolution: Top 5 VRChat Music Concerts, Videos, and Performances. https://news.viverse.com/post/exploring-the-vr-music-revolution-top-5-vrchat-music-concerts-and-performances. [Accessed: 25-May-2026].

[41] LSPosed Framework. https://github.com/lsposed/lsposed. [Accessed: 25-May-2026].

[42] LLDB Debugger. https://lldb.llvm.org/. [Accessed: 25-May-2026].

[43] MITM Proxy. https://www.mitmproxy.org/. [Accessed: 25-May-2026].

[44] PlagueVRC AntiRip. https://github.com/PlagueVRC/AntiRip. [Accessed: 25-May-2026].

[45] Valve Anti-Cheat (VAC). https://help.steampowered.com/en/faqs/view/571A-97DA-70E9-FF74. [Accessed: 25-May-2026].

[46] Play Integrity API. https://developer.android.com/google/play/integrity/overview. [Accessed: 25-May-2026].

[47] Meta Quest: Platform Integrity Attestation API. https://developers.meta.com/horizon/documentation/unity/ps-attestation-api/. [Accessed: 25-May-2026].

[48] Comparing Unity vs Unreal for VR, MR or AR Development Projects. https://xrbootcamp.com/unity-vs-unreal-engine-for-xr-development/. [Accessed: 25-May-2026].

[49] Unreal Engine. https://www.unrealengine.com/en-US. [Accessed: 25-May-2026].

[50] Roblox as a strategic growth platform: what developers need to know. https://newzoo.com/resources/blog/what-developers-need-to-know-about-roblox-in-2025. [Accessed: 25-May-2026].

[51] Meta Horizon Worlds Mobile Focus. https://developers.meta.com/horizon/blog/2026-vr-state-of-the-union-horizon-mobile-focus/. [Accessed: 24-May-2026].

[52] frida-stealth. https://github.com/AsenOsen/frida-stealth. [Accessed: 25-May-2026].

[53] ZygiskFrida. https://github.com/lico-n/ZygiskFrida. [Accessed: 25-May-2026].

[54] android-framework-jar-patching. https://github.com/AsenOsen/android-framework-jar-patching. [Accessed: 25-May-2026].

[55] Zygisk-Il2CppDumper. https://github.com/Perfare/Zygisk-Il2CppDumper [Accessed: 25-May-2026].

[56] frida-il2cpp-bridge. https://github.com/vfsfitvnm/frida-il2cpp-bridge [Accessed: 25-May-2026].

[57] dnSpy: .NET Debugger and Assembly Editor. https://github.com/dnSpy/dnSpy [Accessed: 25-May-2026].

[58] SB Abhinaya, Aafaq Sabir, and Anupam Das. Enabling developers, protecting users: Investigating harassment and safety in {VR}. In *USENIX Security Symposium*, 2024.

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

[60] Mutahar Ali and Habiba Farrukh. Relay and betray: Exploiting client-side authority in multi-user mixed reality, 2026. URL: https://doi.org/10.5281/zenodo.20434355.

[61] Lindsay Blackwell, Nicole Ellison, Natasha Elliott-Deflo, and Raz Schwartz. Harassment in social virtual reality: Challenges for platform governance. *Proceedings of the ACM on human-computer interaction*, 2019.

[62] Qijia Chen, Seyed Mahed Mousavi, Giuseppe Riccardi, and Giulio Jacucci. Investigating the use and perception of blocking feature in social virtual reality spaces: A study on discussion forums. *Proceedings of the ACM on Human-Computer Interaction*, 2025.

[63] Minyeop Choi, Gihyuk Ko, and Sang Kil Cha. {BotScreen}: Trust everybody, but cut the aimbots yourself. In *USENIX Security Symposium*, 2023.

[64] Sam Collins, Alex Poulopoulos, Marius Muench, and Tom Chothia. Anti-cheat: Attacks and the effectiveness of client-side defences. In *Workshop on Research on offensive and defensive techniques in the context of Man At The End (MATE) attacks*, 2024.

[65] Mohammed S Elbamby, Cristina Perfecto, Mehdi Bennis, and Klaus Doppler. Toward low-latency and ultra-reliable virtual reality. *IEEE network*, 2018.

[66] Habiba Farrukh, Reham Mohamed, Aniket Nare, Antonio Bianchi, and Z Berkay Celik. {LocIn}: Inferring semantic location from spatial maps in mixed reality. In *USENIX Security Symposium*, 2023.

[67] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. Android security: a survey of issues, malware penetration, and defenses. *IEEE communications surveys & tutorials*, 2014.

[68] Hanyang Guo, Hong-Ning Dai, Xiapu Luo, Zibin Zheng, Gengyang Xu, and Fengliang He. An empirical study on oculus virtual reality applications: Security and privacy perspectives. In *IEEE/ACM International Conference on Software Engineering (ICSE)*, 2024.

[69] Hilda Hadan, Derrick M Wang, Lennart E Nacke, and Leah Zhang-Kennedy. Privacy in immersive extended reality: Exploring user perceptions, concerns, and coping strategies. In *CHI Conference on Human Factors in Computing Systems*, 2024.

[70] Mee Lan Han, Byung Il Kwak, and Huy Kang Kim. Cheating and detection method in massively multiplayer online role-playing game: Systematic literature review. *IEEE Access*, 2022.

[71] Fadia Hasnaoui, Lamia Zohra Mihoubi, Maria Pateraki, and Miloud Bagaa. Relay-based network architectures for collaborative virtual reality applications. In *IEEE Global Communications Conference (GLOBECOM)*, 2022.

[72] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. Rethinking access control and authentication for the home internet of things (IoT). In *USENIX Security Symposium*, 2018.

[73] Muhammad Ibrahim, Abdullah Imran, and Antonio Bianchi. Safetynot: on the usage of the safetynet attestation api in android. In *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2021.

[74] Ismat Jarin, Yu Duan, Rahmadi Trimananda, Hao Cui, Salma Elmalaki, and Athina Markopoulou. Behavr: User identification based on VR sensor data. *Privacy Enhancing Technologies Symposium (PETS)*, 2025.

[75] Abhishek Kumar, Abdul Khader Jilani Saudagar, Ankit Kumar, Yazeed Masaud Alkhrijah, and Linesh Raja. Innovating medical education using a cost effective and scalable vr platform with ai-driven haptics. *Nature Scientific Reports*, 2025.

[76] Jinghuai Lin, Johrine Cronjé, Carolin Wienrich, Paul Pauli, and Marc Erich Latoschik. Visual indicators representing avatars' authenticity in social virtual reality and their impacts on perceived trustworthiness. *IEEE transactions on visualization and computer graphics*, 2023.

[77] Chen Ling, Utkucan Balci, Jeremy Blackburn, and Gianluca Stringhini. A first look at zoombombing. In *IEEE Symposium on Security and Privacy (SP)*, 2021.

[78] Shiqing Luo, Anh Nguyen, Hafsa Farooq, Kun Sun, and Zhisheng Yan. Eavesdropping on controller acoustic emanation for keystroke inference attack in virtual reality. In *Network and Distributed System Security (NDSS) Symposium*, 2024.

[79] Yan Meng, Yuxia Zhan, Jiachun Li, Suguo Du, Haojin Zhu, and Xuemin Shen. De-anonymizing avatars in virtual reality: Attacks and countermeasures. *IEEE Transactions on Mobile Computing*, 2024.

[80] Andrea Mengascini, Ryan Aurelio, and Giancarlo Pellegrino. The big brother's new playground: Unmasking the illusion of privacy in web metaverses from a malicious user's perspective. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024.

[81] Vivek Nair, Wenbo Guo, Justus Mattern, Rui Wang, James F. O'Brien, Louis Rosenberg, and Dawn Song. Unique identification of 50,000+ virtual reality users from head & hand motion data. In *USENIX Security Symposium*, 2023.

[82] Vivek Nair, Christian Rack, Wenbo Guo, Rui Wang, Shuixian Li, Brandon Huang, Atticus Cull, James F. O'Brien, Marc Erich Latoschik, Louis Rosenberg, and Dawn Song. Inferring private personal attributes of virtual reality users from head and hand motion data. *IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, 2024.

[83] Seonghyun Park, Adil Ahmad, and Byoungyoung Lee. Blackmirror: Preventing wallhacks in 3d online FPS games. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[84] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert Van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *USENIX Security Symposium*, 2004.

[85] Abhinaya SB, Abhishri Agrawal, Yaxing Yao, Yixin Zou, and Anupam Das. " what are they gonna do with my data?": Privacy expectations, concerns, and behaviors in virtual reality. *Privacy Enhancing Technologies Symposium (PETS)*, 2025.

[86] Kelsea Schulenberg, Guo Freeman, Lingyuan Li, and Catherine Barwulor. "creepy towards my avatar body, creepy towards my body": How women experience and manage harassment risks in social virtual reality. *Proceedings of the ACM on Human-Computer Interaction*, 2023.

[87] Carter Slocum, Yicheng Zhang, Nael Abu-Ghazaleh, and Jiasi Chen. Going through the motions:{AR/VR} keylogging from user head motions. In *USENIX Security Symposium*, 2023.

[88] Carter Slocum, Yicheng Zhang, Erfan Shayegani, Pedram Zaree, Nael Abu-Ghazaleh, and Jiasi Chen. That doesn't go there: Attacks on shared state in multi-user augmented reality applications. In *USENIX Security Symposium*, 2024.

[89] Seonghun Son, Chandrika Mukherjee, Reham Mohamed Aburas, Berk Gulmezoglu, and Z Berkay Celik. Side-channel inference of user activities in ar/vr using gpu profiling. 2026.

[90] Zihao Su, Kunlin Cai, Reuben Beeler, Lukas Dresel, Allan Garcia, Ilya Grishchenko, Yuan Tian, Christopher Kruegel, and Giovanni Vigna. Remote keylogging attacks in multi-user VR applications. In *USENIX Security Symposium*, 2024.

[91] Chenxin Sun, Kai Ye, Liangcai Su, Jiayi Zhang, and Chenxiong Qian. Invisibility cloak: Proactive defense against visual game cheating. In *USENIX Security Symposium*, 2024.

[92] Kurt Thomas, Devdatta Akhawe, Michael Bailey, Dan Boneh, Elie Bursztein, Sunny Consolvo, Nicola Dell, Zakir Durumeric, Patrick Gage Kelley, Deepak Kumar, et al. Sok: Hate, harassment, and the changing landscape of online abuse. In *IEEE symposium on security and privacy (SP)*, 2021.

[93] Rahmadi Trimananda, Hieu Le, Hao Cui, Janice Tran Ho, Anastasia Shuba, and Athina Markopoulou. OVRseen: Auditing network traffic and privacy policies in oculus VR. In *USENIX Security Symposium*, 2022.

[94] Martin Vondr'aček, Ibrahim Baggili, Peter Casey, and Mehdi Mekni. Rise of the metaverse's immersive virtual reality malware and the man-in-the-room attack & defenses. *Computers & Security*, 2022.

[95] Hanqiu Wang, Zihao Zhan, Haoqi Shan, Siqi Dai, Maximilian Panoff, and Shuo Wang. Gazeploit: Remote keystroke inference attack by gaze estimation from avatar views in vr/mr devices. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024.

[96] Xiaoying Wei, Xiaofu Jin, and Mingming Fan. Communication in immersive social virtual reality: A systematic review of 10 years' studies. In *International symposium of Chinese CHI (CHCHI)*, 2022.

[97] Yi Wu, Cong Shi, Tianfang Zhang, Payton Walker, Jian Liu, Nitesh Saxena, and Yingying Chen. Privacy leakage via unrestricted motion-position sensors in the age of virtual reality: A study of snooping typed input on virtual keyboards. In *IEEE Symposium on Security and Privacy (SP)*, 2023.

[98] Ch Xi. Virtual reality/augmented reality white paper. *China Academy of Information and Communications Technology (CAICT)*, 2017.

[99] Zhengkun Ye, Ahmed Tanvir Mahdad, Yan Wang, Cong Shi, Yingying Chen, and Nitesh Saxena. Bpsniff: Continuously surveilling private blood pressure information in the metaverse via unrestricted inbuilt motion sensors. In *IEEE Symposium on Security and Privacy (SP)*, 2025.

[100] Yuxia Zhan, Yan Meng, Lu Zhou, Yichang Xiong, Xiaokuan Zhang, Lichuan Ma, Guoxing Chen, Qingqi Pei, and Haojin Zhu. Vpvet: Vetting privacy policies of virtual reality apps. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024.

[101] Jiayi Zhang, Chenxin Sun, Yue Gu, Qingyu Zhang, Jiayi Lin, Xiaojiang Du, and Chenxiong Qian. Identify as a human does: A pathfinder of next-generation anti-cheat framework for first-person shooter games. *IEEE Transactions on Information Forensics and Security*, 2025.

[102] Kexin Zhang, Edward Glenn Scott Spencer Jr, Abijith Manikandan, Andric Li, Ang Li, Yaxing Yao, and Yuhang Zhao. Inclusive avatar guidelines for people with disabilities: Supporting disability representation in social virtual reality. In *CHI Conference on Human Factors in Computing Systems*, 2025.

[103] Qingxiao Zheng, Shengyang Xu, Lingqing Wang, Yiliu Tang, Rohan C Salvi, Guo Freeman, and Yun Huang. Understanding safety risks and safety design in social vr environments. *Proceedings of the ACM on Human-Computer Interaction*, 2023.

A Multi-User MR Applications

Table 5 shows our compiled list of multi-user MR apps. Popularity metrics are sourced from first-party, third-party, and store-reported data. We do not select apps that are predominantly flat-screen and accessed only by a small fraction of users in MR (e.g., Roblox [50]) or that had announced retirement at the time of selection (e.g., Horizon Worlds [51]). We select open source apps (Overte, Vircadia) to show that our attacks generalize beyond Unity-based apps.

Table 5: Multi-user MR apps surveyed for our analysis, ordered by popularity. ✓ / ✗ indicate whether the app is supported on a platform. Use cases span gaming, social/community, design & collaboration, education & training, and healthcare. Popularity figures reflect publicly reported numbers from first- and third-party and store data. FW = client development framework.

App	Quest	Vive	PC VR	Android	Usecase	Popularity	FW.
Gorilla Tag	✓	✗	✓	✓	Gaming	#1 on Quest; 10M+ users	Unity
Rec Room	✓	✗	✓	✓	Social	150M+ installs; 3M+ MAU	Unity
VRChat	✓	✓	✓	✓	Social	#3 on Quest; 20M+ Steam	Unity
Spatial	✓	✓	✗	✓	Design	2.5M+ spaces; 500K+ Android	Unity
Immersed	✓	✓	✗	✗	Design	1.5M+ users; 4K Quest ratings	Unity
Gravity Sketch	✓	✓	✓	✗	Design	1M+ downloads; 100K+ MAU	Unity
Cluster	✓	✗	✗	✓	Social	500K+ Android installs	Unity
Bigscreen VR	✓	✗	✓	✗	Social	500K+ Steam; 5K+ Quest ratings	Unity
ChilloutVR	✗	✗	✓	✗	Social	200K+ Steam owners	Unity
EngageVR	✓	✓	✓	✓	Education	50K+ Steam; 200+ enterprises	Unity
Arkio	✓	✗	✓	✓	Design	20K+ Steam; 10K+ Android	Unity
3D Organon	✓	✓	✓	✓	Health	20K+ Steam; 10K+ Android	Unity
Alcove	✓	✗	✗	✗	Health	515 Quest ratings	Unity
Innerworld	✓	✗	✗	✓	Health	502 Quest ratings; 5K+ Android	Unity
ShapesXR	✓	✗	✗	✓	Design	390 Quest ratings	Unity
Banter	✓	✗	✓	✗	Social	156 Quest ratings	Unity
Nanome	✓	✓	✓	✗	Health	45 Quest ratings; pharma clients	Unity
VXR Labs	✓	✗	✗	✗	Education	21 Quest ratings	Unity
Overte	✗	✗	✓	✗	Social	Open source; self-hosted	C++
Vircadia	✗	✗	✓	✗	Social	Open source; self-hosted	C++

B Methodology Details

Frida. We patched `Frida` for stealth by removing known symbols (e.g., "frida") from socket and thread names, changing the default port number, and applying system-level injection and framework-level modifications to reduce detectability [52–54]. We also bypassed runtime integrity checks by patching common detectors, including `ptrace`/`TracerPid` checks and `frida-gadget` string scans.

IL2CppDumper. Unity IL2CPP binaries (e.g., `libil2cpp.so`) are typically stripped of symbols. We recover method and field names from Unity's `global-metadata.dat`, which stores runtime type and method information required by the IL2CPP engine. For apps with encrypted metadata files (e.g., Rec Room), we extract symbol information at runtime using Zygisk-IL2CppDumper [55] to dump in-memory artifacts. We process the recovered metadata using `IL2CppDumper`, which produces `dump.cs` (a text file containing classes, methods, and fields), `DummyDll/` (stub assemblies viewable in tools such as dnSpy or ILSpy), `script.json` (metadata for analysis scripts such as Ghidra, IDA, or Binary Ninja), and `il2cpp.h` (header files with type definitions).

Ghidra. We import recovered class layouts, method signatures, and function pointer tables into `Ghidra` [27]. This enables analysis of decompiled IL2CPP code as C-like pseudocode with semantic labels instead of raw memory addresses.

frida-il2cpp-bridge. We perform dynamic hooking with frida-il2cpp-bridge [56], which resolves IL2CPP classes, methods, and fields through the runtime's own metadata API in the live process rather than by statically parsing `global-metadata.dat`. This lets us inspect, intercept, and modify managed (C#) methods and memory at runtime.

dnSpy. For cross-layer analysis between native IL2CPP code and Unity's managed structure, we use the managed stub assemblies (`DummyDll`) produced by `IL2CppDumper` and inspect them in `dnSpy` [57]. This allows us to align event handlers, RPC definitions, and high-level class signatures with native functions recovered in Ghidra, improving our understanding of how user actions map to networking and state updates.

App-integrity protections. Several clients distributed through Google Play are wrapped with anti-tamper protection (e.g., PairIP) that verifies package integrity and aborts under tampering or instrumentation. We remove this wrapper before analysis and re-sign the resulting package for installation on our test devices.

TLS interception. To recover the synchronization protocol, we capture the clients' network traffic and decrypt TLS-protected flows, either by installing a system-level root certificate or, where the client pins its certificate, by bypassing the pin at runtime with Frida [93].

C Artifact Appendix

C.1 Abstract

Our artifact supports the paper's security analysis of client- authoritative multi-user mixed reality (MR) apps. It is divided into two components: a public component and a private component.

C.1.1 Public Component

Publicly archived on Zenodo as two files: *relay-and-betray.zip*, the complete toolchain, and *videos.zip*, the recorded video demonstrations of the attacks. The toolchain has five parts:

1. **Static analysis.** A patched Il2CppDumper plus headless Ghidra scripts that recover class hierarchies, method signatures, and field names from stripped Unity IL2CPP binaries.
2. **Gadget injection.** A Frida gadget-injection toolchain for non-rooted Android-based devices, handling split APK sets, Google PairIP license checks, and `TracerPid` anti-debug checks, plus the device scripts that pull, patch, reinstall, and attach to a target.
3. **Runtime logging.** A Frida framework that logs the live IL2CPP class graph and traces outbound synchronization traffic.
4. **Attack hooks.** The main instrumentation methods used and sanitized example hooks.
5. **Helper scripts.** Helper scripts, a Mono runtime bridge for Unity clients that do not use the IL2CPP backend, and verification scripts that check the static-analysis output.

C.1.2 Private Component

For ethical reasons (responsible disclosure), the operational material that weaponizes these tools (i.e., symbol offsets, ready-to-run per-app hooks, the modified C++ client, and the target binaries) is withheld from the public release, as stated in our paper's Open Science statement. It was shared confidentially with the Artifact Evaluation Committee for the reproducibility assessment. It is available to researchers on request.

C.2 Description & Requirements

C.2.1 Security, privacy, and ethical concerns

Following responsible disclosure, the operational material that could enable misuse (the app-specific hooks, symbol offsets, the modified client, and the target binaries) is withheld from the public release and provided only to the artifact evaluators for reproducibility assessment, so that it cannot be misused before the vendor fixes ship. The work involved no human subjects: all demonstrations used our own test accounts in private sessions. The artifact does not contain any destructive components, so it is safe to run the experiments. The instrumentation is intended for use only against systems and accounts you own or are explicitly authorized to test.

C.2.2 How to access

Public Component. Zenodo concept DOI:

https://doi.org/10.5281/zenodo.20434355

The record publishes two files: `relay-and-betray.zip` (the complete toolchain and a stage-by-stage README) and `videos.zip` (the demonstration videos).

Private component (confidential; shared only with AEC). It contains the app-specific hooks, the target binaries, modified APKs, and sample static analysis outputs. Access was granted to the AEC for the duration of the evaluation; that credential has since been revoked, as this material cannot be released publicly for ethical reasons. Consistent with the paper's Open Science statement, we may make it available to other researchers on request, after verification on a case-by-case basis.

C.2.3 Hardware dependencies

Static analysis (Il2CppDumper, Ghidra) runs on any Windows 11, macOS, or Linux machine with a multi-core CPU, $\geq$ 16,GB RAM, and $\sim$30,GB free disk.

Gadget injection runs on macOS or Linux, and additionally needs `adb` and the target Android-based device attached over USB with developer mode enabled.

Runtime logging and attack hooks require running the application on any supported platform. A VR headset (e.g., Quest, Vive) is optional for apps available on PC VR as they can be launched in desktop mode. (Rendering an MR client in desktop mode needs a GPU with more than 4,GB of VRAM.) A GUI is necessary for reproducing results because every attack is confirmed perceptually. You need a second client on a separate endpoint (e.g., headset, PC, smartphone) to verify the attacks.

C.2.4 Software dependencies

Vanilla Windows 11, macOS, or Linux. `setup.sh` / `setup.ps1` install the pinned open-source dependencies: Frida 17.9.1, Node.js, Python 3.12, JDK 21 (for the bundled Ghidra 11.3.2), and the .NET 6 runtime (for the bundled Il2CppDumper); `adb` is needed only for the device path. Running both clients on a single machine additionally needs a sandboxing tool such as Sandboxie-Plus.

C.2.5 Benchmarks

None.

C.3 Set-up

C.3.1 Installation

1. Download both files from the Zenodo DOI and run the setup script (`setup.sh` on macOS/Linux, `setup.ps1` on Windows) to install all dependencies. On Windows, `setup.ps1 -StaticAnalysis` also prepares the JDK and .NET runtime that the bundled Ghidra and Il2CppDumper need.
2. Install the target app and create two free test accounts (one per client).
3. For (E3) only: obtain the private component as described in § C.2.2.

The README walks through the pipeline as ordered stages, each one command with the output to expect: install, obtain the app binary, recover symbols, decompile, verify, log at runtime, and run a hook.

It also works one sanitized example hook end to end, showing how the runtime logger supplies the values that were blanked out.

C.3.2 Basic Test

`frida -version` prints `17.9.1`, and running the dumper on an input binary produces a non-empty `dump.cs` (recovered classes). Together these confirm that both halves of the toolchain work.

C.4 Evaluation workflow

C.4.1 Major Claims

(C1) The static-analysis pipeline deterministically recovers the class, method, and field symbols the attacks rely on, from a stripped Unity IL2CPP binary, with no headset required. This runs with the public component alone. The input binary is supplied by the user, extracted from their own installed app as the README explains, or taken from the sample inputs in the private component; reference output is included there for direct comparison. Proven by experiment (E1).

(C2) Using the operational hooks in the private component against a live app, each of the five invariant violations—perceptual secrecy, state integrity, safety enforcement, availability, and identity integrity—is reproducible across select apps. The attacks need no headset and run on a standard Windows machine with any commercial GPU (more than 4 GB of VRAM). Proven by experiments (E2) and (E3).

We deliberately bound what C2 covers to limit the specialized hardware needed to execute and verify the attacks: the hooks we share target apps that are available across multiple platforms and whose modified client runs on a standard Windows machine. Because these apps' code keeps evolving—and so do the hooks, especially as we coordinate fixes for the disclosed vulnerabilities—and because each attack on each app requires perceptual verification, we provide hooks and step-by-step instructions for a selected set of apps that establish the paper's key claims, rather than for all 20.

C.4.2 Experiments

(E1): *Symbol recovery and decompilation* [∼15 human-min + ∼2 compute-hours]. Run the dumper on the input binaries (`libil2cpp.so` and `global-metadata.dat`), then import the recovered symbols in headless Ghidra, which applies them and exports decompiled C per class. The result is a non-empty symbol dump (`dump.cs`) listing recovered classes, methods, and fields, and a per-class export in which functions carry real names rather than `FUN_xxxx`. Il2CppDumper is deterministic, so the same binary yields the same dump every run; Ghidra is not, as its decompiler makes heuristic choices about naming and typing, so spelling and formatting vary between runs while the recovered structure and logic stay comparable. Verification scripts (`verify.ps1`, `verify.sh`) check both stages and print PASS/FAIL; the output can also be inspected in the Ghidra GUI, or compared against the reference output included in the private component.

(E2): *Runtime logging* [∼15 human-min]. Attach the Frida logger to the running client to enumerate its class graph and locate the methods each attack targets. The printed inventory also identifies the networking and voice SDK in use and the outbound event codes, which are how the placeholders in the sanitized public hooks are filled in.

(E3): *Attack confirmation* [∼2 human-hours]. Launch an attacker client and a victim client in the same private room, making the victim the room host for the host-control attacks. For each attack, load the runtime bridge first, then the corresponding hook; the attach helper does both and identifies the attacker instance automatically. Each attack produces its stated effect on the victim: perception leaking past secrecy settings, unauthorized state or object changes, a non-host exercising host-only controls, a forced crash or disconnection, or impersonation under another user's identity. The demonstration videos in `videos.zip` show the expected outcome for each attack.

C.5 Notes on Reusability

The public component is written to be reused against apps beyond the 20 we studied. The static-analysis pipeline and the runtime logger take no app-specific input: given any Unity IL2CPP client, they recover its symbol map and print its live class graph, networking SDK, and synchronization traffic. The gadget-injection toolchain handles split APKs, license checks, and anti-debug checks generically, so it applies to non-rooted Quest and Vive targets in general. Not every Unity MR client uses the IL2CPP backend, and frida-il2cpp-bridge cannot attach to the ones that do not, so we also ship a Mono runtime bridge exposing an equivalent API; the same hooks therefore apply to either scripting backend. The instrumentation methods in `hook_patterns.js`—modify arguments, modify the return value, read or write fields, and override method logic—are the primitives all of our attacks are built from, and the sanitized examples show how each invariant violation is expressed in those terms. Applying the invariant-guided methodology to a new or updated MR app therefore does not require anything we withheld.

C.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.