Omniscience for the Masses: New Threats in the Metaverse's Democratized World Creation

Andrea Mengascini
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
andrea.mengascini@cispa.de

Ryan Aurelio
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
ryan.aurelio@cispa.de

Jason Polakis
University of Illinois Chicago
Chicago, Illinois, USA
polakis@uic.edu

Giancarlo Pellegrino
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
pellegrino@cispa.de

Abstract

Metaverse platforms increasingly derive their success from user-generated *virtual worlds*: self-contained social and interactive environments, which can be created by *any* ordinary user and scale to billions of visits. Platforms such as Roblox, Horizon Worlds, and VRChat now host millions of creator-built worlds that govern how users see, hear, and interact with one another. While this model enables rapid growth and creativity, it fundamentally delegates control over social interactions and world behavior to untrusted users. In this paper, we present the first systematic security and privacy assessment of metaverse world creators. We survey 25 platforms that support user-created worlds and analyze their world-creation capabilities. Guided by this analysis, we design and implement five novel attacks that exploit creator-provided tools to violate spatial, visual, and auditory constraints in immersive environments, enabling covert user surveillance and manipulation without software vulnerabilities or developer-level privileges. We further show that five previously-proposed attacks can be replicated using only standard world-creation features. Finally, we find that existing platform vetting, runtime protections, and creator policies are insufficient to mitigate malicious world-creator behavior, revealing a fundamental mismatch between users' privacy expectations and the powers granted to world creators.

CCS Concepts

• **Security and privacy** → **Social aspects of security and privacy**; **Privacy protections**.

Keywords

metaverse security, virtual reality privacy, world creator, user-generated content, surveillance attacks, social VR, platform security analysis, immersive environments

This work is licensed under a Creative Commons Attribution 4.0 International License.
CCS '26, The Hague, Netherlands
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3846573

ACM Reference Format:
Andrea Mengascini, Ryan Aurelio, Jason Polakis, and Giancarlo Pellegrino. 2026. Omniscience for the Masses: New Threats in the Metaverse's Democratized World Creation. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3846573

1 Introduction

Metaverses are virtual online environments that allow users to interact, play games, and communicate in an immersive manner, attracting hundreds of millions of daily users [56, 62]. A key driver behind their popularity is the growing support for user-generated content, which has recently further evolved to allow users to create and share entirely new *virtual worlds*. Nowadays, popular platforms such as Roblox, Horizon Worlds, and VRChat derive much of their value from millions of creator-built worlds [56], where individual worlds can accumulate tens of billions of visits [14].

User-created virtual worlds are self-contained experiences such as social gatherings, business meetings, mini-games, or event spaces that users can join within a metaverse platform. Much like Discord channels or IRC rooms, these customizable spaces are executed within the platform's infrastructure, inheriting the platform's runtime and interaction model so creators can build upon them without the need to start from scratch. To define a world's behavior and game logic, creators leverage a range of tools offered by the platforms, from standalone game development environments to in-app WYSIWYG editors supporting visual programming. This shift toward metaverses with world-creation capabilities has introduced a new role in the ecosystem that has not yet been studied: the *world creator*, whose design decisions directly affect the experience of a vast number of users.

The security and privacy of metaverses have been studied extensively in recent years, typically under a threat model that distinguishes between developers and users. Malicious developers, with their privileged access, can exploit VR device sensors to identify, profile, and track users [46, 47] or modify scene-orientation to hijack physical movements and cause harm [6]. Malicious users, by contrast, are assumed to have limited influence over the environment and instead rely on client-side vulnerabilities, memory inspection, or network traffic analysis to track [38], surveil [38], and monitor others [76]. World creators do not fit either category. While they remain ordinary, unverified users from the platform's

perspective, they are entrusted with fine-grained control over the runtime logic and interaction mechanisms of shared social environments. This hybrid role places world creators in a blind spot of existing threat models: they lack developer privileges, yet wield far greater influence over others' experiences compared to traditional users, introducing significant risks.

These security and privacy risks are amplified by the very design choice that makes metaverses feel immersive: platforms emulate real-world physical constraints such as users' sight being blocked by virtual walls and doors, and spatial audio decaying with distance. These cues shape not only how users move and interact (e.g., one cannot walk through a wall), but also how they reason about privacy within virtual spaces (e.g., conversations held behind closed doors are presumed to be private) [34, 96]. Prior work shows that when such cues are respected, users disclose freely; when they are covertly violated, users feel betrayed and exposed [78]. World creators operate precisely at this boundary. The tools that allow them to construct secluded rooms or muted zones can be creatively misused to *stealthily* access, observe, and manipulate avatar state and communication streams within those spaces. The mismatch between user expectation and capability—*the assumption of physical-world-like privacy versus the actual powers of creators*—renders the security and privacy implications of the world-creator role both urgent and critical.

In this paper, we present the first systematic investigation of the security and privacy threats posed by metaverse world creators. As world-creation features are relatively recent and unevenly documented, we begin with a systematic survey of 25 metaverse platforms, including Horizon Worlds, Roblox, and VRChat, that support user-created worlds, and categorize their world-creation capabilities based on the expressiveness of their creation tools. Motivated by our findings and the features provided by the platforms, we conduct an in-depth investigation of five popular metaverse platforms. Therein, we design and implement five novel attacks that exploit creator-provided capabilities to violate spatial, visual, and auditory constraints, with the ultimate goal of achieving *omniscience* within our world. These attacks enable a malicious creator to covertly observe users, monitor private conversations, and manipulate what users see and hear, without relying on software vulnerabilities or developer-level privileges. We further show that five previously proposed metaverse attacks can be replicated using only standard world-creation tools, greatly lowering the barrier for adversarial behavior.

Our study shows that metaverse platforms introduce significant systemic security and privacy risks by delegating extensive control over virtual worlds to ordinary users. We show that world creators can carry out a broad range of attacks, including covert surveillance and user monitoring, using only standard world-creation tools, without exploiting software vulnerabilities. These attacks remain feasible across platforms and tooling models, as attackers can adapt their strategies regardless of whether worlds are built using simplified visual editors or advanced scripting interfaces. Moreover, we find that existing platform defenses are largely ineffective: world vetting processes focus on content rather than behavior, runtime permission systems and warnings are absent or easily bypassed, and users receive no indication when privacy-violating logic is present. Overall, our research demonstrates that any user can readily assume the role of a world creator and deploy malicious worlds at scale, while platforms currently lack meaningful safeguards to protect users from the significant risks inherent to this model.

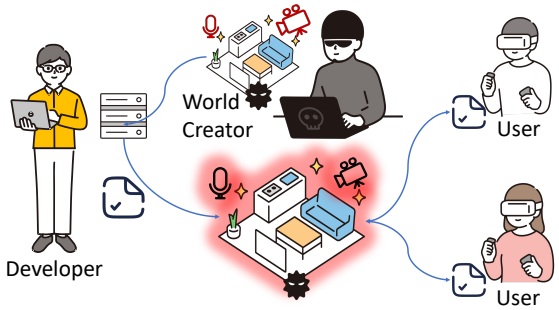


Fig. 1: Threat model and actors in a metaverse platform.

In summary, we make the following contributions:

- Provide the first systematic security and privacy assessment of metaverse world creators, including a taxonomy of 25 platforms and their world-creation capabilities.
- Design and implement five novel attacks that exploit world-creator capabilities to violate spatial and sensory constraints, enabling user surveillance and manipulation.
- Show that five previously proposed metaverse attacks can be realized using only platform-provided world-creation tools, without software vulnerabilities or developer privileges.
- Evaluate platform vetting, runtime protections, and creator policies, and find them insufficient to mitigate malicious world-creator behavior.

2 Threat Model

We now define the threat model underlying our study. Our focus is on privacy and security violations that arise from the legitimate design capabilities delegated to metaverse world creators, rather than from software vulnerabilities or platform misconfigurations. In particular, we study how a malicious creator can violate users' expectations of spatial, visual, and auditory privacy within a creator-controlled world.

Privacy expectations and existing trust models. Our threat model assumes benign users with privacy expectations shaped by two sources: the immersive cues presented by the platform, and prior experience with other multi-user communication systems.

Prior work shows that perceived enclosure in virtual environments encourages self-disclosure, as users feel protected by virtual boundaries such as walls or doors [78]. Metaverse platforms reinforce these expectations through visual occlusion and spatial audio, including distance-based attenuation and, recently, material-aware sound propagation [58]. Users consequently treat these cues as implicit guarantees of exclusivity: opaque barriers are assumed to block visual observation, and conversations that fade with distance are presumed inaudible to outsiders [3, 34, 96]. Users actively rely on features such as privacy zones and lockable rooms as safeguards,

and report strong feelings of exposure and betrayal when these spatial metaphors are silently violated [3, 37]. Crucially, users do not assume these guarantees as absolute: they understand that a host may technically bypass them. What they expect, however, is that such access leaves a trace. Recent user studies found that even technically educated users who acknowledge an administrator could override privacy features still expect this to be signalled rather than silent, and treat covert listening as the main ethical concern [3].

This expectation is not unique to immersive environments; it mirrors the trust model of platforms that precede social metaverse systems, such as Slack, Microsoft Teams, and Zoom. In each, administrators control membership and configuration, but do not have default, invisible access to private conversations. Microsoft Teams' private channels are accessible only to their owners and members; even a team administrator cannot access a private channel unless added as a member [79]. Zoom hosts can join breakout rooms, but they do so as visible participants rather than being able to silently monitor all rooms at once [101]. Chat history export, discovery, and compliance mechanisms exist on some platforms, but they are treated as exceptional and tightly governed [69, 70]. The lesson users carry over is consistent: whoever organizes the interaction space is either visible when present, or has to clear a high, auditable bar to access private exchanges.

As such, it is evident that users find covert, unsignalled surveillance by whoever configured the space neither expectable nor acceptable. Our threat model therefore assumes users expect no world creators—ordinary users who configure and publish worlds—to have monitoring capabilities beyond visible, in-world participation.

Adversary: malicious world creators. We consider the adversary to be a malicious world creator, as illustrated in Fig. 1. Any regular platform user can become a world creator through a straightforward process, typically requiring no additional identity verification beyond account creation. For example, Horizon Worlds allows users to directly enter a creation mode and publish worlds, while platforms such as Roblox and VRChat rely on external editors that integrate seamlessly with user accounts. Although VRChat requires users to meet activity-based "Trust Rank" criteria to publish public worlds, this requirement remains accessible and does not meaningfully prevent malicious participation.

We assume that the attacker can create multiple user accounts, use the official creation tools to publish worlds in the metaverse platform catalogue, and optionally join their own worlds as a regular participant. Similar to prior work in other domains (e.g., analyzing phishing pages [98] or malicious browser extensions [31]), our focus is *not* on how users are lured into entering such worlds, but on what a creator-controlled world can do once users join it.

Attacker capabilities and action space. The attacker's capabilities are strictly confined to those allowed by official world-creation tools. Metaverse platforms provide editors, either client-integrated or standalone, that allow creators to design geometry, upload assets, configure materials, and define interaction logic via scripting or visual tools. Such editors range from accessibility-oriented editors with limited logic primitives to advanced SDKs atop full-featured game engines (e.g., Unity with C# scripting).

The malicious logic considered in this paper is implemented entirely using these legitimate tools. The world creator does not exploit software vulnerabilities, such as memory corruption, improper authorization, or unintended API access. Creators do not have access to server-side databases, low-level networking interfaces, or platform-wide state. Their influence is limited to the worlds and instances they create or control.

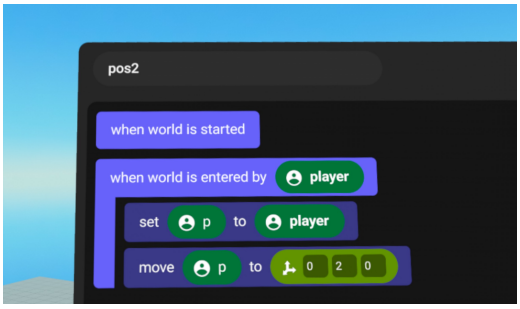


Fig. 2: No-code editor (code blocks) in Horizon Worlds.

For our empirical exploration of novel attacks, we assume an attacker whose objective is *covert omniscience*: the ability to covertly observe all users in a world visually and aurally, regardless of spatial constraints that normally limit perception, as we will see in §4. This is analogous to a communication-platform administrator being able to silently access private exchanges. Beyond these novel attacks, the same creator capabilities substantially lower the technical barrier for previously proposed attacks which required client-side tampering or developer-level access. For our re-implementation of these attacks, we adopt the objectives defined in prior work (as detailed in §5).

3 World Creation Tools

First, we survey and analyze the world-creation toolchains of popular metaverse platforms. We (i) derive a classification of creator toolkits, and use it to select representative platforms for our empirical evaluation, and then (ii) operationalize the primitives exposed to creators into a set of technical capabilities that ground our novel attack designs (§4), and our reproduction of prior attacks (§5).

3.1 Overview of Creation Tools

We identified metaverse platforms that support user world creation via a survey of search engines, VR app stores, and third-party databases. Using strict inclusion criteria (VR support, multiplayer access, general availability, and world-creation functionality), we compiled an initial set of 25 candidates. Following common practice in web/mobile ecosystem studies [18, 100], we restricted the study to platforms that are freely accessible to users. Two researchers independently and iteratively reviewed developer documentation, tutorials, and user manuals for all 25 platforms, explored the creator tooling (apps/editors) of 19, and fully categorized 14, extracting a common set of high-level creation primitives (editor type, logic expression, and creator-facing APIs) and reconciling disagreements. The complete results of our survey are in the extended version of this paper [39], whereas the summary of the platforms relevant for the security analysis of this paper are in Tab. 1.

Editor models. Across all platforms, we observe three recurring editor models. Some platforms build on established game engines by

	Editor			Logic			
Metaverse	Game	In-App	Cust.	No-code	Code	Lang	**Scale (metric + source)**
Roblox [57]			●		●	Lua	150M DAU, 3.5M creators [56]
Horizon Worlds [44]		●	●	●	●	TS	300k users (2022) [28]
VRChat [90]	●			●	●	C#	100k avg. concurrent users [10]
Spatial [73]	●				●	C#	1M+ creators [72]
Frame [22]		●	◐	●	◐	JS	500k global users [23]

Table 1: Creation toolchains for the five evaluated platforms, ● if provided. ◐: Frame dropped support for its Custom Editor (JS) after our experiments. DAU: daily active users.

providing platform-specific SDKs (typically for Unity), allowing creators to develop content externally and upload worlds through platform tooling. Others integrate creation directly into the metaverse client via in-app editors, enabling rapid in situ iteration. Finally, some platforms offer custom external editors, standalone desktop or web-based "studio" tools, that provide a tailored workflow without relying on a mainstream engine.

Logic development. Orthogonal to the editor model, platforms differ in how creators express world logic. Some rely on no-code visual interfaces exposing a fixed set of primitives (e.g., block-based scripting in Fig. 2), while others provide textual scripting or programming languages (e.g., C#, Lua, or JavaScript), offering greater expressiveness and direct support for complex behaviors.

3.2 Creation Capabilities

Next, we identify features exposed to creators through platform-provided creation tools. We focus on those that can be misused to violate users' privacy expectations in virtual environments by subverting architectural or sensorial cues (e.g., unauthorized and covert audio or video surveillance). We also identify the technical primitives required to mount previously reported attacks from the literature (see §5).

A detailed analysis of all 25 surveyed platforms is infeasible, as per-platform capability in-depth analysis is labor-intensive. We therefore select five platforms that are representative of the surveyed set, covering the dominant combinations of tool models and logic-coding paradigms (see Tab. 1). The full selection methodology, inclusion and exclusion criteria, and the complete taxonomy are reported in the extended version of this paper [39] and our open-science materials.

1) Audio. Platforms expose markedly different levels of control over users' audio streams. At one extreme, Roblox enables flexible, programmable audio routing, allowing creators to connect arbitrary audio inputs (e.g., user microphones) to arbitrary outputs (e.g., in-world speakers), effectively decoupling speech from avatar proximity. At the other extreme, Frame, Spatial, and Horizon Worlds strictly bind audio to avatar position and distance, preventing creators from accessing, redirecting, or otherwise manipulating audio beyond default spatial attenuation. VRChat is in an intermediate position: audio remains tied to the speaker's avatar, but creators can adjust parameters such as hearing range and volume, enabling selective amplification or suppression of audibility.

2) Video. Access to virtual camera objects also spans a broad spectrum. Roblox provides the most expressive model: creators can programmatically create and arbitrarily position cameras, and stream their feeds to in-world displays for persistent observation from any virtual viewpoint. VRChat and Spatial support custom, movable cameras but require feeds to be rendered onto visible in-world surfaces. Horizon Worlds allows cameras to attach to invisible, movable entities via scripting, but only on web and mobile clients (not in VR). Frame adopts the most restrictive design: custom cameras are unsupported, and creators are limited to either the fixed third-person browser camera or the first-person VR one that cannot be scripted or repositioned independently of the user's avatar.

3) Custom objects. All five platforms allow creators to import custom 3D assets. At the time of our study, all evaluated platforms supported scripting and allowed creators to modify object visibility, behavior, and interaction logic at runtime, enabling reactive environments. After our study, Frame dropped support for scripting, as discussed in §7.3.

4) Network access. The ability to transmit data outside the platform enables attacks involving logging, tracking, or exfiltration, and platforms impose widely varying restrictions on this capability. Horizon Worlds disallows outbound network requests entirely. VRChat permits limited HTTP GET requests to external resources, restricted to allowlisted domains by default; users may optionally enable an "Allow Untrusted URLs" setting, but POST requests are unsupported. Spatial allows HTTP requests only in public worlds or with a $100/month business subscription and explicitly blocks requests to Spatial-owned domains. Roblox disables HTTP requests by default, but creators can enable them via world settings; requests are processed server-side and rate-limited (500 req/min). At the time of our study, Frame is the most permissive: because worlds execute in the client browser, creators can issue unrestricted HTTP GET and POST requests from client-side scripts.

5) Object attributes: read and write. Modifying non-positional object attributes, such as texture, transparency, or visibility, allows for customizing the design of the virtual environment. All platform editors support reading and writing such attributes. Horizon Worlds is a partial exception: its block-building editor lacks native support for transparent or semi-transparent textures, though creators can achieve transparency by adjusting alpha values (including negative values) or by using the Desktop Editor.

6) Spatial pose data (SPD): read and write. All platforms maintain spatial pose data (position and orientation) for users and objects. Reading and writing this data is universally supported across creation tools, as it underpins core gameplay and interaction features. Horizon Worlds is the sole exception: modifying user pose data requires enabling a world-level setting, which the editor claims triggers a user warning (we did not observe this in practice). Editing object pose data remains universally supported.

7) User identifiable information (UII). Platforms differ in how persistently world creators can identify users across sessions. Frame exposes usernames and, with explicit user consent, registration email addresses. Roblox provides immutable user IDs and usernames, although usernames can be changed for a fee (≈ $17 USD). Horizon Worlds and VRChat expose usernames but limit how frequently they can be changed (once every six months and every

90 days, respectively). Spatial exposes the richest identifier set, including immutable user IDs, usernames, and display names, with usernames and display names freely editable by users.

8) Continued execution. Many attacks (e.g., long-term tracking, fingerprinting) require logic that executes continuously over extended periods. All evaluated creation tools support this through looping constructs or frame-based callbacks, enabling creators to deploy scripts that run indefinitely within their worlds.

3.3 Takeaways

We surveyed 25 metaverse platforms and distilled their world-creation tools into a set of recurring design patterns, which we instantiate through five representative platforms. Across these, world-creation tools expose a rich and largely uniform set of low-level primitives (e.g., persistent execution, access to spatial state, object and attribute control, and user identifiers) that are sufficient to assemble privacy-invasive behaviors. While only a few actions are categorically forbidden, most safeguards constrain specific implementations rather than underlying capabilities, leaving semantically equivalent alternatives available, as we demonstrate in §4 and §5 by presenting concrete attacks. As a result, these restrictions act primarily as friction: by recombining legitimate features, creators can systematically undermine users' spatial and sensory privacy expectations without exploiting vulnerabilities or requiring developer privileges. These primitives form a strict subset of those available to platform developers: creators cannot reach the OS or device-level APIs that enable, e.g., side-channel or keystroke attacks, but they inherit the application's multi-user network model, which exposes the state of all co-present users. However, as we will see, this narrower toolset already suffices to achieve omniscience. The following sections concretely instantiate these building blocks into both novel and previously-proposed attacks.

4 Omniscience Attacks

We now present novel attacks that demonstrate the inherent threat posed by the world creator role emerging in metaverse platforms. We focus on devising new techniques that allow the attacker to achieve omniscience within the worlds they create, by reaching a state of total audio and video surveillance without the introduction of any auditory or visual artifacts that could potentially alert users about an ongoing attack. At heart, our attacks build on the audio and video capabilities offered by the world creator tools, focusing on two separate modalities:

(1) **Superhuman senses.** As the world creator, the attacker has enhanced capabilities regarding the manipulation of their own, or others', audio and video signals.
(2) **Perceptual illusions.** World creation allows the attacker to create environmental illusions that target specific users or widely apply to all other users, and manipulate their perception of the surrounding environment and its properties.

In the following subsections we present our attack techniques, and provide additional information about their internal workings and their implementation in specific metaverse platforms. The attacks are summarized in Table 2. Video demonstrations of all attacks are available in Appendix A, with textual explanations and timestamps for each phase.

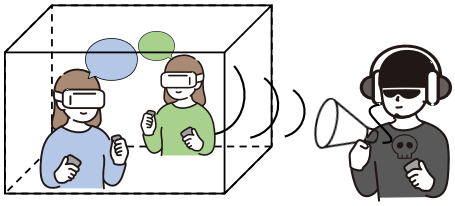


Fig. 3: The parabolic microphone attack.

Attack design. We note that our attacks *do not* require the presence of vulnerabilities, nor do we attempt to uncover any during our empirical analysis. We design our novel attacks using the creator tools as intended; in cases where we do not have direct access to a specific data source, we explore whether we can achieve the objective by designing "malicious" world logic. Ultimately, our attacks demonstrate that *creative* usage of world creation tools allows for the deployment of powerful attacks with significant privacy implications for users. Moreover, we note that while the concrete implementation of each attack can vary across platforms depending on the APIs, objects, and scripting capabilities they provide, the overall design remains consistent. As such, our attacks can be straightforwardly implemented in other platforms.

Experimental setup. We created three accounts in each of the five metaverse platforms, one as creator and the others as victims. Since we had two Oculus Quest devices (Oculus Quest 2 and 3), the third account entered the environment either through the web or mobile client, depending on platform support. We selected the attacker's client (VR or non-VR) based on which option maximized the success and ease of the attacks, using specific clients whenever they exposed additional information or functionalities. All attacks were developed and executed using a Lenovo ThinkPad P1 Gen 6 with an Intel i9-13900H with 32 GB of RAM and a GPU RTX4090 mobile running Windows 11 Pro. We chose the platforms' option to build and compile a world for multiple platforms, e.g., web, mobile, and VR. All attacks were evaluated in a controlled laboratory setting and underwent multi-researcher validation to ensure correctness and reproducibility. We note that for all of our attacks, when audio surveillance is involved users' speech is captured without any noticeable loss or deterioration, and video monitoring clearly captures user avatars and actions.

4.1 Parabolic Microphone

This targeted eavesdropping attack (Fig. 3) allows an attacker to listen to conversations within a metaverse world, regardless of spatial distance or the presence of physical barriers (e.g., walls). The attack relies on audio manipulation capabilities available only to world creators, through platform-provided APIs, to alter how sound is transmitted and perceived. Specifically, the attacker can amplify or re-route audio from users, making conversations audible only to them, without affecting how other participants experience sound in the environment.

VRChat. This platform provides audio functions to the creator that can be used to modify how they hear other users in the world. We used two functions to implement our attack: `VRCPlayerApi.SetVoiceGain` and

	Capabilities								Platform						
Attack	Audio	Video	Cust. Obj.	Obj. Attr.	Network	SPD	UII	Cnt'd Exec.	VRChat	Roblox	Spatial	Frame	Worlds	**Automated**	**Impact**
Omniscience Attacks															
Parabolic Microphone	★			★		★			●	●				◆	Monitors audio of remote users' conversations.
Control Room		★		★		★			●	●	●			◆	Monitors video across multiple private locations.
Astral Projection		★				★			●	●	●	●	●	◆	Monitors spaces via attacker-controlled invisible camera.
Unidirectional Material			★	★					●	●	●	●	●	◆	Monitors users through one-way walls or floors.
Conversation Hijacking	★						★		●	●					Replaces target user's audio to manipulate conversations.
State-of-the-Art Attacks															
User Re-identification [46]					★	★	★		◐	●	◐	●	◐	◆	Tracks users persistently across sessions or worlds.
Social Logging [85]					★	★	★	★	◐	●	◐	●	◐	◆	Profiles users by recording interactions and behaviors.
User DoS [38]						★	★	★	●	●		⊘	◐	◆	Disrupts user participation by blocking or isolating them.
FP with Motion Data [49]					★	★	★	★	◐	●	◐	⊘	●	◆	Tracks and profiles users based on movement patterns.
Human Joystick [6]						★	★	★	●	◐	●	⊘	◐	◆	Manipulates user's movement or view covertly.

Table 2: Required capabilities and feasibility of each attack across metaverse platforms. ★: Capability required for the attack; ●: Feasible with platform capabilities; ◐: Feasible with alternative implementation; ⊘: Not feasible in Frame anymore, due to scripting support being discontinued after the completion of our experiments. ◆ : Automated; ◆: Automated under conditions.

`VRCPlayerApi.SetVoiceDistanceFar`. The first function adjusts a specific player's volume, and the second sets the maximum distance for hearing that player's voice; these are executed on the client side. The script runs only when the local player's name matches the attacker's name, ensuring that the changes only affect how the attacker hears others.

Using the provided functions we can choose which players we want to hear. We created a script that first checks whether the local player is the attacker. If that is the case, we set `VRCPlayerApi.SetVoiceDistanceFar` to the maximum value for all players, allowing the attacker to hear everyone from any distance. Then, we created a UI object inside the world containing a list of players and a slider for controlling each player's volume. We also attached the slider to a script that uses `VRCPlayerApi.SetVoiceGain` based on the value of the slider. Using this method, the attacker can choose which players to hear by adjusting the voice sliders accordingly, allowing them to focus on specific conversations.

Roblox. The Roblox Audio API enables developers to route and manipulate audio using modular components such as `AudioListener` (microphone), `AudioEmitter` (speaker), and `Wire` to connect them. Leveraging these primitives, we implemented a parabolic microphone attack that simulates a wiretapped environment. Our setup consists of hidden "microphone rooms" containing an `AudioListener` object, and a separate "speaker room" accessible only to the attacker, containing an `AudioEmitter`. By linking these components with a `Wire`, we reroute real-time voice data to the attacker's location, bypassing audio attenuation and occlusion due to environmental aspects (e.g., walls or distance). This acts as a directional microphone, capturing conversations remotely and replaying them elsewhere, undetectable by victims or other users.

Other platforms. The attack could not be replicated on the other platforms due to the lack of appropriate audio APIs.

4.2 Control Room

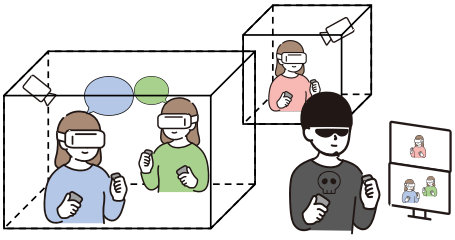


Fig. 4: Control Room for surveillance.

Audio eavesdropping attacks capture speech but miss the visual interactions central to metaverse experiences, such as gestures, movement, and object use. To observe this activity, we developed a video-based surveillance attack (Fig. 4) that allows passive monitoring across multiple regions of the world. In this attack, the adversary creates a private "control room" containing a wall of screens, each displaying the live feed from invisible cameras placed throughout the world. The cameras are strategically positioned in common or sensitive areas to monitor user behavior without their knowledge (e.g., board room meeting). To execute the attack, the platform must support the creation of camera objects and the ability to redirect their visual output to in-world display surfaces.

VRChat and Spatial. Both platforms allow an attacker to place multiple `Camera` objects throughout the virtual world and display each camera's view on a `Plane` object that acts as a monitor. The `Camera` objects are invisible by default, allowing the attacker to observe users without their knowledge. In VRChat, this video monitoring can be combined with parabolic microphones to also capture conversations, providing a comprehensive view of user activity.

Roblox. Roblox does not provide a direct way to stream a camera's view onto a screen. Instead, the attacker can use a virtual `Camera` with a `ViewportFrame`—a UI object that renders 3D objects inside its bounds [64]—to simulate CCTV surveillance. Invisible cameras are placed in key areas, and a script cycles through all objects each frame to determine what should appear in the camera's view, then renders this to the screen. The attacker can view these feeds in a private control room or an attacker-only visible HUD using a `ScreenGui`. Combined with a parabolic microphone, this enables real-time video and audio monitoring across the world.

Other platforms. This attack is not feasible on platforms that do not expose camera and screen-like objects to world creators, or that restrict the ability to capture and replicate user or object movement within the scene.

4.3 Astral Projection

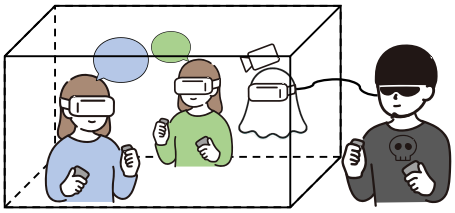


Fig. 5: The astral projection attack.

This attack allows the attacker to decouple their viewpoint from their avatar, effectively rendering them invisible to others while freely navigating their world (Fig. 5). The attacker controls a virtual camera independent of their in-world position, enabling them to monitor private spaces and interactions. The attack requirements are lower than those of the control room attack, as it does not require in-world screens, only to move the attacker's own camera away from its avatar. Algorithm 1 details the implementation. A is the world creator who later joins as a participant; upon joining, A gains access to the live player list for ID-based target selection, then decouples the camera to surveil V via key input or automatic position tracking.

Roblox. The `CameraType` property allows full programmatic control of the player camera. By setting the camera to `Scriptable`, it can be moved throughout the environment using custom controls, regardless of the avatar's position. The attacker's audio input also follows the camera, so they can hear conversations from the camera's current location, enabling both visual and audio surveillance.

VRChat and Spatial. Here it is not possible to independently move the actual player camera. Instead, the attacker creates a new camera object in the world, which is attached to a screen-like object that only the attacker can see. The attacker can move and rotate this camera anywhere in the environment with keyboard or controller inputs (`Input.GetKey`, `transform.Translate`). This allows for visual surveillance, but the camera does not capture audio. To monitor conversations, the attacker must combine this attack with an audio-based technique, such as the parabolic microphone.

Horizon Worlds. Creators can script cameras by attaching them to invisible, movable objects, using the "Fixed Camera Position with Entity" option via the `playercamera` entity. Camera movement is controlled through custom input buttons that reposition the underlying object, enabling the attacker to navigate the environment independently of their avatar. To use this feature, the attacker must connect through either the web or the mobile client, as VR clients remain limited to first-person view [16]. While this attack allows full visual surveillance, it does not capture spatial audio; audio monitoring requires combining a separate technique.

Algorithm 1 Astral Projection. A (world creator) moves their camera near V, observing V from any angle while invisible to all users.

On world join *(A only, platform-specific)*:
Roblox: *cam.CameraType* ← SCRIPTABLE # hijack player cam
VRChat / Spatial: *cam* ← CAMERA(*pos*) # invisible world obj
Horizon: *cam* ← CAMERA(*pos*); CAM.ATTACH(*entity*)

Attacker selects target V *(ID-based)*:
V ← GETPLAYERBYID(*target_id*) # V unaware of selection
CAMERA.POSITION.SET(V.*position*) # camera to V's location

Attacker observes V *(covert, persistent)*:
while *true* **do**
  Δ ← *manual* ? READINPUT : V.*position* − *cam.position*
  CAMERA.POSITION.SET(*cam.position* + Δ)
end while

FrameVR. FrameVR does not support custom or scriptable cameras. However, users accessing the world via a web browser in third-person mode can zoom out and pan the camera nearly without restriction, observing most of the environment from a distance. The camera orientation remains fixed on the avatar and cannot be rotated freely, so the attacker can see who is present and what they are doing, but not always from all angles. This technique provides only visual monitoring; to eavesdrop on conversations, attackers would need to combine this with an audio-based attack.

4.4 Unidirectionally Transparent Material

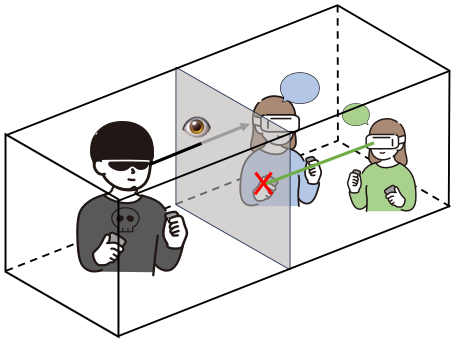


Fig. 6: Unidirectionally Transparent Material.

This attack employs the perceptual illusion modality to build an environment that creates the illusion of a private environment for users (Fig. 6). In more detail, this attack relies on a carefully designed space inspired by the properties of interrogation rooms

(as seen in movies). Essentially, the attacker can employ a variety of different designs to create a complex of rooms, where parts of the room (i.e., floor, walls, ceiling) are comprised of a material that is one-way see-through. This allows the attacker to place themselves behind a wall, or under the floor, or on the roof, of a room designed to spy on users. Using a unidirectionally transparent object, the attacker can see everything in the victim's room through this object. However, other users perceive that object as solid and therefore can not see what is behind it. The attacker can also hear the players in the room if they are within hearing range. As the world creator, the attacker can determine the room size, ensuring it falls within their hearing range. Alternatively, the attacker can combine this attack with any of the audio-based attacks. The only requirement for this attack is for the platform to support the creation of custom objects.

All platforms. Since all platforms in our study allow uploading custom 3D assets, we created a one-way see-through object using Blender [20]. To achieve this effect, we enabled the `Backface Culling` option in the material shader settings and selected the `Camera` flag. This configuration makes the object renderable from only one side, appearing opaque on one side while transparent from the other, creating a unidirectional window. We used this to construct spy rooms that visually deceive users in all platforms.

Platforms with scripting support. Additionally, on platforms that support user scripting (i.e., VRChat, Roblox, Spatial, Horizon Worlds), we also implemented an ad-hoc, identity-based wall-visibility control mechanism. We attach a script to the wall object that checks the local user's username at runtime. By default, the object is enabled, but if the script detects the attacker's name, it disables the object for them. For all other users the wall appears solid, while for the attacker it is invisible, allowing direct observation without needing custom shaders. This approach can complement or replace custom object-based transparency.

4.5 Conversation Hijacking

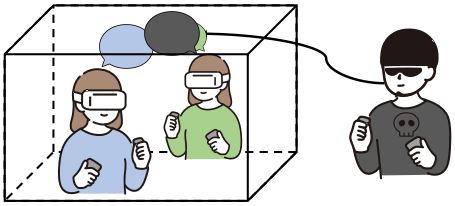


Fig. 7: The conversation hijacking attack.

While the previous attacks are passive and focus on monitoring users, we have also designed an attack that allows the attacker to actively interfere with ongoing user interactions. This enables a wide range of attack scenarios, including spreading misinformation, malvertising, influencing the outcome of a collaborative decision, or manipulating users' opinions. Specifically, the attacker impersonates another user by muting the target player's audio and increasing their own volume for the target's conversation partners. In other words, the attacker is able to replace the target users' audio feed with an audio stream of their choice (Fig. 7). This attack requires the ability to modify other players' audio settings.

To strengthen the attack, the adversary can pre-record messages using a cloned version of the victim's voice or leverage near-instant voice cloning tools for real-time impersonation. Recent work shows that convincing voice clones can be generated from just a few seconds of audio [52], making such attacks increasingly practical.

VRChat. Consider user A (victim) and user B (conversation partner). The attacker uses a script to adjust user B's hearing of nearby players. First, the attacker mutes user A by setting their audio distance to zero for user B, using `VRCPlayerApi.SetVoiceDistanceFar`. Then, the attacker boosts their own volume to the maximum, making their voice heard by user B. As a result, user B only hears the attacker, believing they are still communicating with user A. Algorithm 2 details the implementation of the conversation hijacking attack for VRChat, where V and other users are unaware of the manipulation.

Roblox. Here, the attacker achieves a similar effect by modifying the `Muted` property of user A's `AudioDeviceInput` (microphone) object, effectively silencing them. Simultaneously, the attacker uses an `AudioEmitter` (speaker) to inject their own voice into the environment, targeting user B. To preserve the illusion of authenticity, the attacker can script a fake "speaking" icon to appear next to user A's name, mimicking the platform's native UI indicator for speaking. This visual manipulation further deceives the victim into believing the impersonation is genuine.

Other platforms. This attack is not possible on platforms that do not allow the world creator to modify other users' audio settings.

Algorithm 2 Conversation Hijacking (VRChat). A silences V for all users and injects their own voice into B's feed, making B believe they are still talking to V. A then restores the original audio state.

On world start *(attacker only)*:
1: *players* ← GetAllPlayers
2: **for each** $p \in$ *players*: add entry with toggles Mute(p), Relay(p)
3: Update panel on player join / leave

Attacker activates hijack (selects victim V, bystander B):
4: SetVoiceDistanceFar(V, 0) # silence V globally
5: Sync Mute(V) and Relay(B) state to all clients
6: **if** localPlayer = B **then** # change only B's state
7: SetVoiceDistanceFar(A, ∞) # B hears A instead of V
8: **end if**

4.6 Attack Automation and Stealthiness

Next, we characterize our attacks by their degree of automation, i.e., whether they can persist as world logic after deployment or require continued attacker intervention. This characterization complements our platform feasibility analysis, summarized in Tab. 2.

Parabolic Microphone and *Control Room* are automated across all platforms where we implemented them: once configured, their logic is embedded in the world and enables unattended A/V surveillance.

Other attacks expose a spectrum of automation depending on whether the attacker monitors fixed locations or dynamically follows specific users. *Astral Projection* and *Unidirectionally Transparent Material* can be fully automated when monitoring static areas (e.g., a private room), by placing a camera or the creator's viewpoint

at a predetermined position. *Unidirectionally Transparent Material* remains automatable in targeted scenarios by positioning the creator avatar in a concealed vantage point (e.g., below a one-way floor) and tracking the victim. Similarly, *Astral Projection* supports target-following automation on all platforms except FrameVR, where the camera cannot be locked onto a user and must be manually repositioned. In both attacks, however, when the attacker wishes to switch targets to adapt to unfolding context, manual intervention is needed, making both partially automated.

For automated and conditionally automated attacks, data collection is straightforward using standard client-side screen and audio recording tools; we use OBS [51] for web clients, and built-in recording on Meta Quest or mobile devices.

Finally, *Conversation Hijacking* is not automatable in practice. The attack requires real-time semantic understanding and precise timing of audio manipulation to impersonate a specific user in an ongoing interaction, which current platforms do not support programmatically, leaving such attacks reliant on human intervention.

Stealthiness. Our attacks introduce no auditory or visual artifacts that would alert users to ongoing surveillance or interference. In surveillance attacks, audio remains spatialized normally, and no visible indicators reveal attacker-controlled cameras, concealed observation points, or rerouted audio paths. Similarly, *Conversation Hijacking* alters how select users perceive a conversation without observable cues for the impersonated victim or surrounding participants. Extensive testing confirmed that victim accounts perceive no differences during attack execution compared to benign worlds. We measured the client-side overhead our attacks impose on the victim, repeating each measurement three times per platform, except on Spatial, which dropped support during our study. The impact is negligible: on average, frame rate changes by -1.0% and GPU utilization by -1.5%, while CPU utilization rises by only $\sim 10\%$. We report the full per-attack, per-platform breakdown in the extended version of this paper [39].

5 Revisiting State-of-the-Art Attacks

Next, we analyze prior attacks targeting metaverse and VR platforms. Unlike the omniscience attacks of §4, these are reproductions of prior-literature attacks. We show that previously-proposed attacks, originally requiring either (i) a considerable level of technical expertise [38], (ii) the presence of specific software vulnerabilities [85], or (iii) developer-level privileges [6], can be straightforwardly realized using only platform-provided world creation tools. These attacks' technical requirements align with the world-creation capabilities discussed in §3.2, which we use for the attacks' re-implementation.

To ground our analysis in real-world threats, we surveyed 60 academic papers on metaverse and VR security, identifying 32 that propose attacks. We grouped these into seven categories and focused on four: sensitive data collection, denial of service, fingerprinting, and physical manipulation. Attack categories relying on user deception, such as social engineering or clickjacking, or on system-level access, such as keylogging, were excluded from our analysis. The extended version of this paper [39] summarizes the survey, including categories, representative works, and inclusion criteria. Below, we briefly present the selected attack categories and corresponding attacks we implemented.

Sensitive data collection. Both developers and users can retrieve sensitive data about other users from virtual environments. For example, developers can use their privileged access to collect and analyze user positions, and hand and head movements [48], e.g., to infer a user's physical space [83]. Users can collect data by tampering with the client-side execution environment [38]. Since this is the most prevalent category in our survey, we implement two representative attacks: user re-identification and social logging.

Denial of Service (DoS). DoS attacks aim to disrupt the users' experience or virtual environment functionalities. For example, both developers [7] and users [38] can encapsulate target objects with other objects, making them inaccessible.

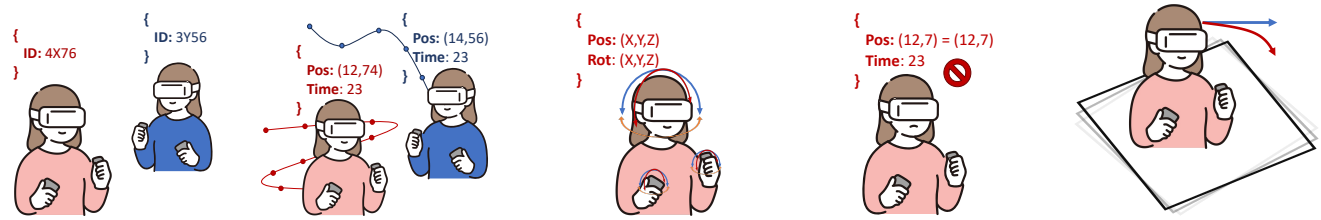
Fingerprinting. Developers can use movement patterns or device characteristics [46, 49] to track users across sessions, by creating persistent user profiles without the users' knowledge or consent. Accordingly, we will demonstrate how we can read and exfiltrate a user's motion data.

Physical world. These attacks exploit the virtual environment to disrupt the user's physical experience or manipulate their actions. For example, attackers can manipulate object appearances to redirect the user's gaze [53], mismatch virtual and physical movements to alter walking paths [6], or induce frame drops and erratic motion to cause cybersickness [81]. We implement the human joystick attack [6] from this category.

5.1 Proof-of-Concept Attacks

We implement five representative state-of-the-art attacks (Fig. 8) across the selected platforms, to demonstrate how world creators can trivially implement attacks from all major categories, without requiring sophisticated technical skills or specific vulnerabilities, as was the case in the past. On Roblox, we reproduced several of these attacks with great ease using the platform's built-in AI coding assistant, which generated and attached the required scripts from a short prompt in a few attempts, without questioning their intent.

User re-identification attack. This attack links users across sessions or worlds using persistent identifiers, and requires two capabilities: access to user-identifiable information and a channel for data exfiltration. Frame's no-code editor allows creators to embed user-specific information (e.g., emails or usernames) into predefined request templates. This should trigger user consent prompts, but we found a method to access identifiers directly via mesh objects without any warnings. Data can then be exfiltrated through client-side scripts using the `Fetch` function. Roblox exposes immutable user IDs and mutable usernames; creators can exfiltrate them using built-in HTTP requests. Horizon Worlds blocks network access, so we implement a hidden control room (see Fig. 9) that displays usernames on a canvas; the creator can join the world and use Oculus casting with Selenium and OCR to extract data from rendered frames. VRChat also restricts outbound requests to static allowlisted URLs, so we log data locally on the client when a known username (the attacker) is detected and extract it from our disk. In Spatial, identifiers are accessible via the `actorService` API; while HTTP access requires a $100/month subscription, logs



(a) Re-identification. (b) Social Logging. (c) FP with Motion Data. (d) User DoS. (e) Human Joystick.

Fig. 8: Illustrative scenarios for state-of-the-art metaverse attacks.

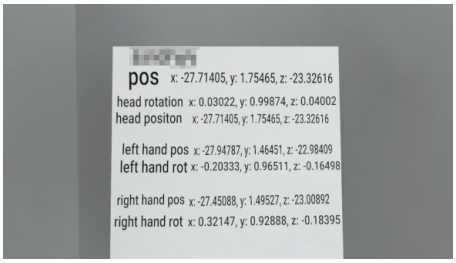


Fig. 9: Control room terminal in Horizon Worlds, Personal Identifier blurred for anonymization.

can be written to the browser console and scraped using Selenium, bypassing the paywall.

Fingerprinting through motion data. This attack builds motion profiles from users' head and hand movements and requires access to spatial pose data, identifiable user information, network access, and continuous execution. All platforms expose positional data for user avatars. For example, in Frame, while HMD and controller positions are poorly documented, we used `GetAllMeshes` to identify the relevant meshes and extract position and rotation. Other platforms provide explicit APIs, such as `leftHand.position` in Roblox or `GetTrackingData(Head)` in VRChat. For user identifiers and exfiltration, we reused the techniques described in the re-identification attack. To ensure continuous data collection, we relied on per-frame callbacks or looping constructs; e.g., `while(true)` in Roblox, `OnBeforeRenderLoop()` in Frame, and the visual `while` block in Horizon Worlds.

Social logging. This attack involves monitoring users' presence and spatial movements within a virtual world to infer social behaviors. It requires continuous spatial data access, network communication capabilities, and continuous execution of the logging script. To access all players, we iterate over a parent object containing user entities. Roblox, VRChat, and Spatial provide dedicated methods: `GetPlayers()` , `VRCPlayerApi.GetPlayers()` , and `actorService.actors` , respectively. In Horizon Worlds, we built a user list by handling user join and leave events. In Frame, client-side scripts start logging automatically upon world load, so no iteration is needed.

Human joystick. This attack manipulates a user's position or view by gradually adjusting their camera or movement direction, causing them to unknowingly follow a physical path. The goal is to disorient users and influence real-world movement without their awareness. It requires write access to camera or positional data, and runs continuously to apply subtle, incremental changes over time. In Frame, Spatial, and Roblox, we directly modified the camera rotation via `GetMeshByName(..).cameras[0].rotation`, `avatar.rotation`, and `CurrentCamera.CFrame`, respectively. In VRChat, we used `TeleportTo(pos, rot)` to control rotation without altering position. Horizon Worlds blocks direct pose manipulation unless "Enable Player Movement" is activated, which should warn users, though we could not trigger the warning. To bypass this, we scripted a collidable platform tracking the user's feet and moving accordingly, affecting position but not rotation. This adaptation allowed for human joystick behavior but is limited in scalability and precise directional control. For continuous execution, we used the same techniques as the fingerprinting attack.

User DoS. This attack disrupts a user's experience by immobilizing them or repeatedly teleporting them to inaccessible locations, requiring identification of the target, control over their movement, and continuous execution. In VRChat, we use the built-in `Immobilize()` method. In Roblox and Spatial, we repeatedly teleport the user to a fixed vector using `humanoidRootPart.Position` and `avatar.position`. In Frame, we modify the parent node of the active camera's position, effectively teleporting the user. Additionally, we freeze the client with an infinite loop (`while(true)`), halting all input. Horizon Worlds does not allow direct pose changes unless "Enable Player Movement" is enabled (with an untriggered warning); as an adaptation, we use collidable objects to manipulate the player's position, catapulting them into the sky or out of bounds. User targeting and execution persistence follow the same approaches as prior attacks.

6 Platform Protections

Having shown that world creators can achieve omniscient visibility and control within their own worlds using only standard creation tools, we now examine whether metaverse platforms provide effective safeguards against such threats, or meaningful signals to affected users. This section evaluates the protections platforms claim to offer, spanning vetting processes, in-world safety mechanisms, runtime permissions and warnings, and creator-facing policies, and weigh these claims with our empirical observations.

Vetting and review processes. Several platforms describe some form of review prior to, or following, the publication of worlds. To understand the scope of such processes, we reviewed the public documentation of 14 platforms (see the extended version of this

paper [39]), focusing on whether, and how, world content is evaluated before being made accessible to other users. Where review mechanisms are described, they appear primarily oriented toward moderating visible or thematic content. For example, VRChat's Community Labs review addresses inappropriate or offensive material [89]; Roblox requires creators to complete a "Maturity & Compliance Questionnaire" centered on violence, fear, or crude humor [61]; and Horizon Worlds applies content ratings and policy checks aimed at adult or harmful content [43]. Community discussions similarly frame these systems as content moderation tools rather than as mechanisms for assessing world logic or technical behavior [5, 9].

Across the entire duration of the experiments described in this paper, we did not observe *any* intervention, review feedback, or additional scrutiny related to the technical behaviors implemented in our worlds. From our perspective as creators and participants, publishing and interacting with these worlds did not surface any signals suggesting that creator logic was being examined beyond content-level considerations. While this does not rule out the existence of internal checks, it suggests that any such processes are not externally visible and, at minimum, do not manifest in ways that would constrain or deter the behaviors of our attacks.

In-world social safety mechanisms. In addition to vetting, platforms emphasize in-world safety features intended to give users control over their immediate social experience. Platforms such as VRChat [91] and Horizon Worlds [40] allow users to mute or block others, hide avatars, or reduce visual and audio effects, which are primarily designed to mitigate harassment or unwanted user-to-user interactions. These controls operate at the level of interpersonal interaction and do not constrain the behavior of world creators or the logic embedded in a world. Consequently, they do not affect the attacks presented in this paper, which are implemented through creator-controlled world logic rather than direct user behavior.

Permissions and user awareness. A more direct approach to constraining world-creator behavior would involve runtime permission systems or user-facing warnings shown when sensitive capabilities are exercised. Among the platforms we evaluated, only *two* explicitly reference such mechanisms. FrameVR presents a consent dialog when a creator attempts to access a user's email address via its no-code action system; however, we found that the same information is also accessible via JavaScript, allowing creators to bypass this warning entirely. Horizon Worlds requires creators to enable a world-level setting to manipulate player movement, and the editor indicates that enabling this option will notify users [39]; however, we did not observe any corresponding user-facing warning in practice. For the remaining platforms, we found no indication of permission prompts or warnings related to creator access to sensitive capabilities.

In our experiments, as creators we accessed and combined sensitive inputs, such as spatial pose data, audio routing, and persistent user identifiers, without triggering permission prompts or warnings for affected users. In cases where platforms claim that opt-in settings or notifications should be displayed, we did not observe them in practice. Consequently, users in our attack scenarios received no indication that their behavior was being monitored, logged, or manipulated beyond what is shown in a typical world experience. Our observations are limited to externally visible behavior and do not exclude the possibility of undocumented mechanisms. Nonetheless, from a user's perspective, these systems did not provide actionable transparency or awareness during our evaluation.

Creator policies and terms. Finally, metaverse platforms articulate expectations for creator behavior through policies, terms of service, and SDK or analytics guidelines. To assess how these documents address the behaviors enabled by our attacks, we reviewed publicly available creator-facing policies for the five analyzed platforms and mapped explicit restrictions to the techniques and outcomes of our attacks.

In general, policies are most explicit in restricting the collection of sensitive or real-identity information, profiling or cross-service tracking, and the export of user-derived data off-platform. Platforms such as Meta, Roblox, and VRChat prohibit in-world collection of certain identifying data [41, 42, 60, 63, 93]. Profiling is prohibited on Roblox and conditionally permitted on other platforms, typically contingent on notice or consent [59, 71, 93]. Exporting user data is similarly constrained in VRChat's SDK License, Roblox's third-party data policies, and Frame's terms [24, 59, 87].

At the same time, these policies coexist with creation tools that continue to expose the technical primitives required to implement the restricted behaviors. Enforcement appears to rely primarily on terms-of-service obligations and retrospective action, rather than on preventative technical controls. As a result, responsibility for compliance is delegated to creators, while users have limited visibility into how worlds handle their data or shape their experience.

7 Discussion

Our work reveals systemic security and privacy risks introduced by the world-creator model adopted by metaverse platforms. Here, we distill cross-cutting lessons, discuss potential mitigations and their challenges, and outline the scope and limitations of our study.

7.1 Lessons Learned

World creation fundamentally changes the trust model. Metaverse platforms blur the traditional boundary between users and developers by delegating powerful, developer-like capabilities to ordinary users acting as world creators. While creators remain untrusted from the platform's perspective, they are granted fine-grained control over runtime logic, sensory cues, and interaction rules. This hybrid role does not fit existing threat models, introducing new attacks that are neither purely developer- nor user-driven. Crucially, crafting this attack logic demands far less skill than full application development, widening the pool of capable adversaries.

World creators violate privacy and trust expectations. Visual occlusion, spatial audio, and architectural metaphors implicitly signal privacy and exclusivity [68, 96], and even when privileged actors override these boundaries, users expect such access to be visible or signalled [3]. This matches popular systems such as Zoom and Slack, where administrators configure spaces but do not silently monitor private exchanges. Our attacks show that creator-built worlds can violate this trust model: ordinary creators can silently observe, listen to, track, and manipulate users through standard creation tools. The core risk is thus not creator control itself, but unobservable creator authority that violates expectations of visible or signalled access.

Omniscience emerges from composition, not vulnerabilities. Individually, most world-creation features appear benign. Collectively, through creative design and synthesis, they enable omniscient surveillance and manipulation. Across platforms and tools, creators can recombine persistent execution, spatial state access, and object control to achieve powerful attacks. Importantly, these capabilities emerge without exploiting software vulnerabilities or undocumented APIs.

The barrier to impactful attacks is low. Compared to prior metaverse attacks that required client tampering or network analysis, the attacks in this paper can be implemented using visual editors or simple scripts. Even when platforms restrict specific APIs, creators can often reconstruct equivalent behaviors through alternative means. Additionally, this barrier is also dropping, as tooling evolves from block-based editors to AI assistants that synthesize attack logic from a prompt. This shifts the threat model from expert attackers to, essentially, any motivated user.

Existing defenses are structurally misaligned. Platform defenses focus on content moderation, user-to-user safety controls, or contractual restrictions. These mechanisms do not meaningfully constrain creator-controlled world logic. As a result, malicious behavior can persist without triggering vetting, runtime warnings, or user awareness, even when it violates stated policies.

7.2 Mitigation Strategies and Open Challenges

Our findings suggest that there is no single, standalone solution to the risks introduced by creator-controlled worlds. Instead, mitigating these threats requires a combination of complementary measures, each with its unique challenges in the metaverse setting.

Raising the entry barrier for creators. One immediate mitigation is to increase the cost of becoming a creator [65]. Other ecosystems, most notably mobile app platforms, rely on account vetting, including government ID verification [11, 15]. Similar mechanisms could limit who can publish public worlds, or what capabilities are unlocked at early stages. However, this directly conflicts with the low-friction, creator-driven growth model that metaverse platforms currently favor.

World vetting beyond visible content. In principle, platforms could vet worlds in a manner analogous to mobile app review, inspecting logic and behavior before publication. In practice, this is substantially harder. Our attacks rely solely on standard features and can be embedded inconspicuously in otherwise benign experiences. Unlike mobile apps, the harmfulness of a world often depends on how logic unfolds dynamically in a 3D, multi-user environment.

Traditional program analysis techniques struggle in this domain. Static analysis cannot capture how spatial, visual, and auditory behaviors interact at runtime. Dynamic analysis would require exploring a vast state space shaped by user movement, interaction, and timing. Even detecting components such as cameras or logging logic is insufficient, as these are common in legitimate worlds.

Permissions and transparency help, but only to a point. Runtime permissions or transparency labels (e.g., indicating audio routing, data collection, or cameras) could improve user awareness. However, many benign features would trigger frequent prompts, risking habituation [2, 82]. Moreover, disclosures often remain under creator control, limiting their effectiveness. Designing meaningful, abuse-resistant permission systems remains an open problem.

Reconsidering platform-enforced invariants. A more fundamental approach may be to reassert certain platform-level invariants, such as limits on sensory access or observation, that creators cannot override. This would represent a shift away from fully malleable worlds toward stronger guarantees of user protection, but would require careful trade-offs with expressiveness and creativity.

7.3 Limitations and Scope

Platform selection. Our study focuses on free-to-access metaverse platforms, where large-scale user activity occurs and aligning with prior security research. Paid or enterprise-oriented metaverses may exhibit different trade-offs, which we leave to future work. We also do not measure the prevalence of malicious worlds in the wild. Understanding how often these attacks occur in practice is an important direction for future work.

Platform updates. Metaverse platforms are rapidly evolving, and creator tooling changed after our study. For example, Meta introduced a desktop editor that supports TypeScript, which in practice lowered the effort required to implement several attacks; importantly, attacks developed using the in-app editor remain feasible under the new tooling. Conversely, Frame deprecated its custom editor and removed general scripting support after our experiments, rendering some script-based attacks from §5.1 infeasible. Nonetheless, while specific attacks may become easier or harder as tooling changes, our core finding *that powerful creator capabilities persist across platforms and editor models* remains unchanged.

8 Related Work

This paper presents the first study on the risks of the world-creation capabilities in metaverse platforms. Here, we discuss prior work on risks of user-generated content in different contexts, as well as prior security analyses of the metaverse through the lens of the well-established roles of developers and users.

User-generated content. A related line of work explores the implications of allowing user-generated content (UGC) to be uploaded or utilized in applications. UGC and creators are often seen as user-empowered agents capable of producing harmful content or environments, particularly in games. Zhang et al. [99] and Kou et al. [32] have explored harmful patterns in UGC, such as embedding problematic incentives like pervasive microtransactions in Roblox. However, these studies focus on harmful design patterns rather than specific attacks executable through the game, which has been widely covered by the press, including themes like sexual content [8] and terrorism [4]. UGC can also serve as an attack vector, enabling malicious actors to actually exploit systems. For instance, file upload validations can be bypassed to upload unrestricted content [33], potentially leading to serious risks such as remote code execution [21]. While traditional attacks often rely on encapsulating payloads in seemingly benign files, the metaverse introduces a unique dimension where the payload is not just a file but the actual logic of a virtual world. This distinction underscores the need to study UGC-based threats in metaverse environments, where malicious logic can use user experiences to launch attacks.

Security threats in the metaverse. As part of our analysis of the risks introduced by the world creator role, we have also demonstrated how previously-proposed attacks can be trivially deployed. Next, we briefly discuss relevant prior work in the space.

Developers. Developers create and maintain the platform applications, typically leveraging game engines such as Unity [92] or Unreal Engine [27], or using custom-built engines [95]. They also manage the supporting infrastructure, including servers, databases, and tools for world creation. Developers possess significant control over the metaverse, granting them control over both the application and its underlying infrastructure. If malicious, developers can conduct various attacks. For example, they can exploit immersive scenarios to redirect users' movements in the physical world [6] or collect sensitive user data such as physical characteristics [46] and movement patterns [47]. As operators of the platform infrastructure, they can gather and analyze extensive user data, potentially enabling surveillance [80]. This threat has led researchers to investigate mitigations, such as differential privacy mechanisms [50].

Users. Users access the metaverse through client applications that render virtual environments and synchronize with servers. Their attacks typically exploit client-server vulnerabilities or overreach by the client app. Common techniques include XSS to bypass security mechanisms [85], client-side memory scanning to capture sensitive information [38], and network tampering to infer personal details [76]. When traditional vulnerabilities are absent, attackers resort to complex methods such as observing hand gestures or gaze patterns to infer information [94, 97]. Recent defenses, such as anti-cheat systems [88] and limiting sensitive information shared among clients [77], aim to reduce users' attack surface.

9 Conclusion

In this paper, we performed the first assessment of the security and privacy risks introduced by metaverse world creators. We showed that ordinary users can create worlds that violate other users' spatial, visual, and auditory privacy, and distort their environmental perception. This is achieved through the creative use of official toolchains for fabricating malicious world logic across platforms, without exploiting software vulnerabilities or developer privileges. We demonstrated the significant threat posed by the world creator role through concrete instantiations of novel attacks that enable covert user surveillance and manipulation, and re-implementations of previously-proposed attacks. Our research further shows that existing defenses are ineffective: vetting focuses on content rather than behavior, permission systems and user-facing warnings are limited or absent, and users remain unaware of invasive logic executing within worlds. Overall, our findings reveal a fundamental mismatch between users' privacy expectations and the powers granted to world creators, highlighting an urgent need for stronger, behavior-aware safeguards in metaverse platforms.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable feedback. This work was supported by the National Science Foundation under grants CNS-2211574 and CNS-2143363. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the NSF.

References

[1] AltspaceVR. 2024. AltspaceVR | Be there, together. https://web.archive.org/web/20210108130656/http://www.altvr.com/.
[2] Bonnie Brinton Anderson, Anthony Vance, C Brock Kirwan, Jeffrey L Jenkins, and David Eargle. 2016. From warning to wallpaper: Why the brain habituates to security warnings and what can be done about it. *Journal of Management Information Systems* 33, 3 (2016), 713–743.
[3] Jakki O. Bailey, Xinyue (Sally) You, Andrea Stevenson Won, Sun Joo (Grace) Ahn, and Blair MacIntyre. 2025. Students' Privacy and Ethical Concerns of Using Social Virtual Worlds for Online Learning. *Proc. ACM Hum.-Comput. Interact.* 9, 7, Article CSCW247 (Oct. 2025), 22 pages. doi:10.1145/3757428
[4] Russell Brandom. 2021. Roblox is struggling to moderate re-creations of mass shootings - The Verge. https://www.theverge.com/2021/8/17/22628624/roblox-moderation-trust-and-safety-terrorist-content-christchurch.
[5] Kerry Breen. 2021. Roblox: Experts, users warn about inappropriate content. https://www.today.com/parents/roblox-experts-users-warn-about-inappropriate-content-t235027.
[6] Peter Casey, Ibrahim Baggili, and Ananya Yarramreddy. 2021. Immersive Virtual Reality Attacks and the Human Joystick. *IEEE Transactions on Dependable and Secure Computing* 18, 2 (March 2021), 550–562. doi:10.1109/TDSC.2019.2907942
[7] Kaiming Cheng, Arkaprabha Bhattacharya, Michelle Lin, Jaewook Lee, Aroosh Kumar, Jeffery F. Tian, Tadayoshi Kohno, and Franziska Roesner. 2024. When the User Is Inside the User Interface: An Empirical Study of UI Security Properties in Augmented Reality. In *33rd USENIX Security Symposium*.
[8] James Clayton and Jasmin Dyer. 2022. Roblox: The children's game with a sex problem. https://www.bbc.com/news/technology-60314572.
[9] Roblox Community. 2025. False DMCA Takedown Against My Game. https://devforum.roblox.com/t/false-dmca-takedown-against-my-game-a-wake-up-call-for-roblox-and-developers/4013020.
[10] VRChat Community. 2026. Metrics - VRChat Community Dashboards. https://metrics.vrchat.community/.
[11] Google Play Console. 2026. Google Play Developer Verification: Required documents. https://support.google.com/googleplay/android-developer/answer/15633622
[12] HTC Corporation. 2024. VIVERSE - Your Portal to the Immersive 3D Internet. https://www.viverse.com/.
[13] Microsoft Corporation. 2024. Introducing Microsoft Mesh | Connect like never before. https://www.microsoft.com/en-us/microsoft-teams/microsoft-mesh.
[14] Roblox Corporation. 2026. Murder Mystery 2. https://www.roblox.com/en/games/142823291/Murder-Mystery-2.
[15] Apple Developer. 2026. Identity verification. https://developer.apple.com/help/account/membership/identity-verification/
[16] Meta Developers. 2025. Camera API Examples Tutorial - Module 1: Setup. https://developers.meta.com/horizon-worlds/learn/documentation/tutorial-worlds/camera-api-examples-tutorial/module-1-setup.
[17] DigiGods. 2024. DigiGods - Play for FREE on Meta Quest! https://www.digigods.gg/.
[18] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. 2011. A Study of Android Application Security. In *20th USENIX Security Symposium*.
[19] Overte e.V. 2024. About — Overte. https://overte.org/.
[20] Blender Foundation. 2026. blender.org - Home of the Blender project - Free and Open 3D Creation Software. https://www.blender.org/.
[21] OWASP Foundation. 2024. Unrestricted File Upload. https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload.
[22] Frame. 2024. Frame. https://framevr.io/.
[23] Frame. 2026. Immersive 3D Collaboration Spaces. https://learn.framevr.io/.
[24] FrameVR. 2024. Frame Terms of Service. https://learn.framevr.io/terms.
[25] Grab Games. 2024. GRAB. https://grabvr.quest/.
[26] Trass Games. 2024. Yeeps: Hide and Seek – Gods of Gravity. https://godsofgravityvr.com/pages/yeeps-hide-and-seek.
[27] Hadean. 2024. What Epic Games' State of Unreal 2023 means for the Metaverse. https://hadean.com/blog/what-epic-games-state-of-unreal-2023-means-for-the-metaverse/.
[28] Alex Heath. 2022. Meta's social VR platform Horizon Worlds hits 300,000 users - The Verge. https://www.theverge.com/2022/2/17/22939297/meta-social-vr-platform-horizon-300000-users.
[29] Rec Room Inc. 2024. Rec Room. https://recroom.com/.
[30] Alpha Blend Interactive. 2024. ChilloutVR - Free to play multiverse platform. https://abinteractive.net/.
[31] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. 2014. Hulk: Eliciting malicious behavior in browser extensions. In *23rd USENIX Security Symposium*.
[32] Yubo Kou and Xinning Gui. 2023. Harmful Design in the Metaverse and How to Mitigate it: A Case Study of User-Generated Virtual Worlds on Roblox *(DIS '23)*. doi:10.1145/3563657.3595960

[33] Taekjin Lee, Seongil Wi, Suyoung Lee, and Sooel Son. 2020. FUSE: Finding File Upload Bugs via Penetration Testing. *Network and Distributed System Security Symposium*.

[34] Josephus Jasper Limbago, Robin Welsch, Florian Müller, and Mario Di Francesco. 2025. Don't They Really Hear Us? A Design Space for Private Conversations in Social Virtual Reality. *IEEE TVCG* (2025).

[35] Somnium Space Ltd. 2024. Somnium Space. https://somniumspace.com/.

[36] Sine Wave Entertainment Ltd. 2024. Home | sinespace. https://sine.space/.

[37] Christian Mai, Tim Wiltzius, Florian Alt, and Heinrich Hußmann. 2018. Feeling alone in public: investigating the influence of spatial layout on users' VR experience. In *NORIDCHI 18*.

[38] Andrea Mengascini, Ryan Aurelio, and Giancarlo Pellegrino. 2024. The Big Brother's New Playground: Unmasking the Illusion of Privacy in Web Metaverses from a Malicious User's Perspective *(CCS '24)*. doi:10.1145/3658644.3690249

[39] Andrea Mengascini, Ryan Aurelio, Jason Polakis, and Giancarlo Pellegrino. 2026. Omniscience for the Masses: New Threats in the Metaverse's Democratized World Creation (Extended Version). arXiv:2609.12554 [cs.CR] https://arxiv.org/abs/2609.12554

[40] Meta. 2024. Safety and privacy tools for your child or teen in Meta Horizon Worlds. https://www.meta.com/en-gb/help/quest/articles/horizon/safety-and-privacy-in-horizon-worlds/safety-tools-for-your-teen-horizon/.

[41] Meta. 2025. Code of Conduct for Virtual Experiences. https://www.meta.com/de/en/legal/quest/code-of-conduct-for-virtual-experiences/.

[42] Meta. 2025. Quest Games Terms of Service. https://www.meta.com/de/en/legal/quest/games-terms-of-service/.

[43] Meta. 2025. Worlds content guidelines. https://developers.meta.com/horizon-worlds/learn/documentation/save-optimize-and-publish/restrictions-to-worlds-in-horizon.

[44] Inc. Meta Platforms. 2024. Meta Horizon Worlds on Meta Quest | Quest VR games | Meta Store. https://www.meta.com/en-gb/experiences/meta-horizon-worlds/2532035600194083/.

[45] Multiverse. 2024. Multiverse Dev Zone. https://www.multiverseupdates.com/.

[46] Gonzalo Munilla Garrido, Vivek Nair, and Dawn Song. 2024. SoK: Data Privacy in Virtual Reality. *Proceedings on Privacy Enhancing Technologies* 2024, 1 (Jan. 2024), 21–40. doi:10.56553/popets-2024-0003

[47] Vivek Nair, Wenbo Guo, Justus Mattern, Rui Wang, James F. O'Brien, Louis Rosenberg, and Dawn Song. 2023. Unique identification of 50,000+ virtual reality users from head & hand motion data *(SEC '23)*. USENIX Association.

[48] Vivek Nair, Gonzalo Munilla Garrido, Dawn Song, and James O'Brien. 2023. Exploring the Privacy Risks of Adversarial VR Game Design. *Proceedings on Privacy Enhancing Technologies* 2023, 4 (2023). doi:10.56553/popets-2023-0108

[49] Vivek Nair, Christian Rack, Wenbo Guo, Rui Wang, Shuixian Li, Brandon Huang, Atticus Cull, James F. O'Brien, Marc Latoschik, Louis Rosenberg, and Dawn Song. 2024. Inferring Private Personal Attributes of Virtual Reality Users from Ecologically Valid Head and Hand Motion Data. IEEE Computer Society.

[50] Vivek C Nair, Gonzalo Munilla-Garrido, and Dawn Song. 2023. Going Incognito in the Metaverse: Achieving Theoretically Optimal Privacy-Usability Tradeoffs in VR. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*. doi:10.1145/3586183.3606754

[51] OBS. 2026. Open Broadcaster Software. https://obsproject.com/

[52] Zengyi Qin, Wenliang Zhao, Xumin Yu, and Xin Sun. 2023. Openvoice: Versatile instant voice cloning. *arXiv preprint arXiv:2312.01479* (2023).

[53] G. Nikki Ramirez, Jameson Spivack, and Brendan David-John. 2024. Deceptive Patterns and Perceptual Risks in an Eye-Tracked Virtual Reality. In *2024 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. 341–344. doi:10.1109/VRW62533.2024.00068

[54] Remio. 2024. Remio VR. https://www.remiovr.com/.

[55] Resonite. 2024. Resonite. https://resonite.com/.

[56] Roblox. 2024. Metrics & Insights. https://brands.roblox.com/metrics-insights.

[57] Roblox. 2024. Roblox. https://www.roblox.com/home.

[58] Roblox. 2025. [Beta] Acoustic Simulation - Updates / Announcements - Developer Forum. https://devforum.roblox.com/t/beta-acoustic-simulation/3634265.

[59] Roblox. 2025. Creator Third-Party App Policy. https://en.help.roblox.com/hc/en-us/articles/37924211313044-Creator-Third-Party-App-Policy.

[60] Roblox. 2025. Creator Third-Party App Terms. https://en.help.roblox.com/hc/en-us/articles/15887203369620-Creator-Third-Party-App-Terms.

[61] Roblox. 2025. How you can help us make Roblox safer. https://create.roblox.com/docs/safety.

[62] Roblox. 2025. RDC 2025. https://corp.roblox.com/newsroom/2025/09/roblox-rdc-2025.

[63] Roblox. 2025. Roblox Community Standards. https://en.help.roblox.com/hc/en-us/articles/203313410-Roblox-Community-Standards.

[64] Roblox. 2025. ViewportFrame | Documentation - Roblox Creator Hub. https://create.roblox.com/docs/reference/engine/classes/ViewportFrame.

[65] Roblox. 2026. New Publishing Requirements for Games. https://devforum.roblox.com/t/new-publishing-requirements-evaluation-process-for-games/4573166.

[66] Third Room. 2024. Third Room. https://thirdroom.io/landing.

[67] Inc. Sansar. 2024. Sansar | Official Site - The world's leading social virtual reality platform. https://www.sansar.com/.

[68] Abhinaya S.B., Abhishri Agrawal, Yaxing Yao, Yixin Zou, and Anupam Das. 2025. "What are they gonna do with my data?": Privacy Expectations, Concerns, and Behaviors in Virtual Reality. *Proc. Priv. Enhancing Technol.* 2025 (2025).

[69] Slack. 2026. Export your workspace data. https://slack.com/help/articles/201658943-Export-your-workspace-data.

[70] Slack. 2026. Guide to Slack import and export tools. https://slack.com/help/articles/204897248-Guide-to-Slack-import-and-export-tools.

[71] Spatial. 2024. Spatial Privacy Policy. https://www.spatial.io/privacy.

[72] Spatial. 2026. About. https://www.spatial.io/about.

[73] Inc. Spatial Systems. 2024. Spatial - Create Immersive UGC, Virtual Classrooms, Experiential Marketing. https://www.spatial.io/.

[74] Sava Studios. 2024. Penguin Paradise. https://savastudios.xyz/.

[75] Throwback Studios. 2024. DerbyVR. https://throwback.studio/.

[76] Zihao Su, Kunlin Cai, Reuben Beeler, Lukas Dresel, Allan Garcia, Ilya Grishchenko, Yuan Tian, Christopher Kruegel, and Giovanni Vigna. 2024. Remote Keylogging Attacks in Multi-user VR Applications.

[77] Apple Support. 2024. About the security content of visionOS 1.3 - CVE-2024-40865. https://support.apple.com/en-us/120915.

[78] Philipp Sykownik, Divine Maloney, Guo Freeman, and Maic Masuch. 2022. Something personal from the metaverse: goals, topics, and contextual factors of self-disclosure in commercial social VR. In *CHI 2022*.

[79] Microsoft Teams. 2026. IT Admins - Private channels in Microsoft Teams. https://learn.microsoft.com/en-us/microsoftteams/private-channels.

[80] Rahmadi Trimananda, Hieu Le, Hao Cui, Janice Tran Ho, Anastasia Shuba, and Athina Markopoulou. 2022. OVRseen: Auditing Network Traffic and Privacy Policies in Oculus VR. 3789–3806.

[81] Samaikya Valluripally, Aniket Gulhane, Khaza Anuarul Hoque, and Prasad Calyam. 2022. Modeling and Defense of Social Virtual Reality Attacks Inducing Cybersickness. *IEEE Transactions on Dependable and Secure Computing* (2022).

[82] Anthony Vance, David Eargle, Jeffrey L. Jenkins, C. Brock Kirwan, and Bonnie Brinton Anderson. 2019. The Fog of Warnings: How Non-essential Notifications Blur with Security Warnings. In *SOUPS 2019*. USENIX Association.

[83] John Vilk, David Molnar, Benjamin Livshits, Eyal Ofek, Chris Rossbach, Alexander Moshchuk, Helen J. Wang, and Ran Gal. 2015. SurroundWeb: Mitigating Privacy Concerns in a 3D Web Browser. In *2015 IEEE SP*.

[84] Villa. 2024. Metaverse Terraforming Platform. https://www.villa.rocks/.

[85] Martin Vondráček, Ibrahim Baggili, Peter Casey, and Mehdi Mekni. 2023. Rise of the Metaverse's Immersive Virtual Reality Malware and the Man-in-the-Room Attack & Defenses. *Computers & Security* (2023). doi:10.1016/j.cose.2022.102923

[86] Neos VR. 2024. Neos Metaverse. https://neos.com/.

[87] VRChat. 2021. VRChat SDK License (Material License Agreement). https://hello.vrchat.com/legal/sdk.

[88] VRChat. 2022. The VRChat Security Update. https://hello.vrchat.com/blog/vrchat-security-update.

[89] VRChat. 2024. I want to make a world public. https://help.vrchat.com/hc/en-us/articles/360060848014-I-want-to-make-a-world-public.

[90] VRChat. 2024. VRChat. https://hello.vrchat.com/.

[91] VRChat. 2024. VRChat Safety and Trust System. https://docs.vrchat.com/docs/vrchat-safety-and-trust-system.

[92] VRChat. 2024. VRChat Unity 2022.4.1p3. https://docs.vrchat.com/docs/vrchat-202241p3.

[93] VRChat. 2025. VRChat Terms of Service. https://hello.vrchat.com/legal.

[94] Hanqiu Wang, Zihao Zhan, Haoqi Shan, Siqi Dai, Maximilian Panoff, and Shuo Wang. 2024. GAZEploit: Remote Keystroke Inference Attack by Gaze Estimation from Avatar Views in VR/MR Devices *(CCS '24)*. doi:10.1145/3658644.3690285

[95] webaverse studios. 2024. An open metaverse engine for everyone. https://github.com/webaverse-studios/webaverse.

[96] Julie Williamson, Jie Li, Vinoba Vinayagamoorthy, David A Shamma, and Pablo Cesar. 2021. Proxemics and social interactions in an instrumented virtual reality workshop. In *2021 CHI*.

[97] Zhuolin Yang, Zain Sarwar, Iris Hwang, Ronik Bhaskar, Ben Y. Zhao, and Haitao Zheng. 2024. Can virtual reality protect users from keystroke inference attacks? *(SEC '24)*. USENIX Association.

[98] Penghui Zhang, Adam Oest, Haehyun Cho, Zhibo Sun, RC Johnson, Brad Wardman, Shaown Sarker, Alexandros Kapravelos, Tiffany Bao, Ruoyu Wang, et al. 2021. Crawlphish: Large-scale analysis of client-side cloaking techniques in phishing. In *2021 IEEE SP*.

[99] Zinan Zhang, Sam Moradzadeh, Xinning Gui, and Yubo Kou. 2024. Harmful Design in User-Generated Games and its Ethical and Governance Challenges: An Investigation of Design Co-Ideation of Game Creators on Roblox. CHI PLAY (2024). doi:10.1145/3677076

[100] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shomir Wilson, Norman M. Sadeh, Steven M. Bellovin, and Joel R. Reidenberg. 2016. Automated Analysis of Privacy Requirements for Mobile Apps. In *NDSS 2016*.

[101] Zoom. 2026. How to manage breakout rooms in a meeting. https://support.zoom.com/hc/en/article?id=zm_kb&sysparm_article=KB0062540.

A Open Science

Data sharing. We share in our repository the survey data for the platforms, literature survey data on attacks, and the finalized codebooks with resolved conflicts. The full application-survey, the complete coding of the systematic analysis, and the notes on how coding conflicts were resolved are all included in the artifact repository. All data, codebooks, resolution notes, and supporting materials are available in the main artifact repository: https://github.com/andrea-mengascini/worldcreator-artifact/.

Attack artifacts. Video demonstrations of all our attacks are available at https://andrea-mengascini.github.io/worldcreator-artifact/. These videos include clear textual descriptions and timestamps for each phase of the attacks. The same videos are also provided for offline download within the main artifact repository. The repository also includes the 3D model used to implement the one-way material, enabling complete reproduction of this component of our evaluation.

B Ethical Considerations

We carefully considered ethical implications throughout the entire lifecycle of this research, from the design of experiments to the potential impact of the results. In this section, we outline our ethical considerations for each stakeholder involved and the measures taken to mitigate potential risks. Importantly, for ethical reasons, we did not upload any public worlds containing attacks or malicious payloads to test the platform vetting processes. Uploading such worlds could have exposed real users to harm, and therefore we restricted all experimental evaluations to private or unlisted instances under our exclusive control.

Metaverse platform users. We ensured that our experiments would not inadvertently affect the users of the tested metaverse platforms. To achieve this, all tests were conducted in private or unlisted instances. These instances were restricted to our test accounts, ensuring public users could not inadvertently join or be impacted. For platforms supporting private environments, we utilized password-protected or invite-only instances. On platforms that did not provide private instances, we created unlisted instances and conducted all experiments exclusively with our test accounts.

After completing the experiments, we removed all test rooms. In cases where deletion was not possible, we replaced private or unlisted instances with blank worlds stripped of all functionalities to eliminate any potential residual impact.

Metaverse platform providers. We adhered to the intended functionalities by designing and uploading worlds using only official APIs and tools provided by the platforms. Our experiments did not involve testing or exploiting vulnerabilities in the platforms' tools or server infrastructure, ensuring compliance with their operational integrity.

Responsible disclosure. When we identified possible attacks, we initiated a responsible disclosure process with the affected platforms, following the protocol detailed below. For each platform, we first requested a valid security contact point using available communication channels such as web forms, email, or official social media. After obtaining the appropriate contact, we shared a detailed technical report describing the issues and steps to reproduce the attacks. Roblox replied through their bug bounty program, stating that the reported issue does not constitute a security vulnerability according to their criteria. This response further indicates that, in practice, malicious worlds leveraging the techniques detailed in this work would pass the vetting process of major platforms. Meta similarly concluded that the reported behavior does not constitute a security or privacy issue, arguing that a world creator gains no additional information beyond what would be available through in-world proximity. For the remaining platforms, we have notified them and did not receive a response.

Artifact sharing. To reduce the risk of misuse, we will not publicly release executable attack scripts, exploit code, or ready-to-deploy malicious world files while the underlying issues remain unmitigated. We only shared technical artifacts privately with reviewers, affected platform providers and, upon request, we will share them with vetted researchers for research purposes.

C Platform Survey Results

Tab. 3 reports the complete results of our platform survey. It lists the 25 candidate platforms with their editor models and logic paradigms, including the platforms discarded from the study. Highlighted rows mark the platforms selected for our security analysis.

	Editor			Logic		
Metaverse	Game	In-App	Cust.	No-code	Code	Lang
Horizon Worlds [44]		●	●	●	●	TS
Frame [22]		●	◐	●	◐	JS
Resonite [55]		●		●		-
Neos VR [86]		●		●		-
VRChat [90]	●			●	●	C#
Rec Room [29]	●	●		●	●	C#
Spatial [73]	●				●	C#
Viveport Verse [12]	●				●	JS
Sansar [67]	●				●	C#
ChilloutVR [30]	●				●	C#
Sinespace [36]	●				●	Lua
Roblox [57]			●		●	Lua
Overte [19]			●		●	JS
Third Room [66]			●		●	C#

Discarded (in-app editor, logic not assessed): Multiverse [45], Remio [54], GRAB [25], Penguin Paradise [74], DigiGods [17]
Discarded (not evaluated): Microsoft Mesh [13], Somnium Space [35], Yeeps [26], Villa [84], AltspaceVR [1], Derby [75]

Table 3: Categorization of world creation tools; platforms discarded from the study are listed below the table. ◐: Frame dropped their support for the Custom Editor which supported JavaScript after our experiments.