The Big Brother's New Playground: Unmasking the Illusion of Privacy in Web Metaverses from a Malicious User's Perspective

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

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

Abstract

Metaverses are virtual worlds where users can engage in social exchanges, collaborate, or play games. Their clients now are JavaScript programs that run inside modern web browsers. They implement functionalities typical of multiplayer video games, like 3D and physics engines, requiring them to maintain complex data structures of objects in the browser's memory. Unfortunately, these objects can be accessed and manipulated by malicious users, allowing them to learn about events beyond the ones rendered on screen or to hijack the physics of the metaverse to spy on other users.

In this paper, we propose one of the first comprehensive security assessments for web clients of metaverse platforms. We begin with a survey and selection of three metaverse platforms and introduce a software-centric threat modeling approach designed to identify the security-relevant entities. Then, we propose a JavaScript global object snapshot diffing technique to identify in-memory objects correlated with the attribute and design 10 attacks, of which eight successfully executed against at least one of the metaverses, enabling a malicious user to perform audio/video surveillance or continuous user position tracking — to mention a few — who could exacerbate current threats posed by stalkers and online abusers. Finally, we discuss the implications of our attacks should the metaverse become a business tool and possible solutions.

CCS Concepts

• **Security and privacy → Web application security**; Domain-specific security and privacy architectures.

Keywords

Virtual Reality (VR) security and privacy; Metaverse security and privacy; WebXR; immersive web applications

ACM Reference Format:
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. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3690249

CCS '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3690249

1 Introduction

Metaverses are virtual worlds where users can interact, engage in social exchanges, collaborate, or play games. Increasingly, metaverses have begun offering cross-platform web-based clients that are client-side JavaScript programs running inside modern web browsers.

Web clients implement a variety of functionalities typical of multiplayer video games, such as rendering the 3D environments and handling collisions and physics. These functionalities require web clients to maintain complex data structures of objects in the browsers' memory, whose changes are shared between the clients to achieve *global state consistency*. Many objects are for the client's user presence, e.g., the user's avatar position and rotation; the rest are for entities controlled by other users, e.g., their avatars, and the server, e.g., objects in the scene. While we expect that a client can dispatch updates on the objects under its control, the system should prevent the client from sharing updates on other objects. Unfortunately, the objects in the client's memory can be accessed and manipulated by a malicious user using readily available tools, e.g., DevTools, allowing the user to learn about events beyond the ones rendered on screen or to hijack the physics of the metaverse to spy on other users. The impact and extent to which such access and manipulations threaten the security and privacy of users have not been explored to date.

The security and privacy of virtual environments have gathered the attention of the research community primarily due to concerns around cheating in online games (see Yan and Randell [32] for a classification of cheating in games). Cheating is the manipulation of game mechanics to gain an unfair advantage, which manifests in various forms. Notable among these are the use of aimbots to automatically target opponents (see, e.g., [7, 33]) and wallhacks to see through solid objects (see, e.g., [3, 22]), thereby significantly undermining the integrity of the gaming environment. However, the metaverse introduces a shift from conventional online games due to its inherent social component. Unlike traditional games, metaverses support a broader spectrum of interactions, including social gathering [23], virtual concerts [21], and social gaming [15]. This shift lessens the focus on cheating, introducing heightened concerns regarding the security properties that support these social dynamics, such as confidentiality, integrity, and availability. Very recently, several works have begun focusing on these security properties, making a variety of assumptions about potential attackers. These include a server harvesting network-level data [28], a developer fingerprinting users with ad-hoc room designs [13, 20], a user (or a server) fingerprinting users based on their mechanic

hand motion [19], and classical web attacker exploiting XSS vulnerabilities [29]. Unfortunately, none of these studies have comprehensively evaluated the security and privacy of metaverse platforms from the perspective of a malicious user accessing or manipulating in-memory data structures of the client-side program.

In this paper, we propose one of the first comprehensive security assessments for web clients of metaverse platforms. Our security assessment begins with a survey and selection of three metaverse platforms which have hosted events such as conferences and fashion shows. As we aim to deliver an extensive security analysis, we introduce a software-centric threat modeling approach designed to identify the security-relevant entities and attributes of the metaverses, namely, the *Entity-Attribute Model*. Starting from each attribute, we propose a JavaScript global object snapshot diffing technique to identify in-memory objects correlated with the attribute and test the objects for read and write access and the implication of these operations. Finally, we design 10 attacks based on these access patterns, demonstrating the feasibility of eight attacks against at least one platform. Overall, our security assessment shows that current implementations may not be sufficiently robust against malicious users. Several of our attacks enable targeted and untargeted forms of audio/video surveillance and continuous user position tracking, exacerbating the risks posed by various threat actors. Notable among these are stalkers and online abusers, who could leverage these vulnerabilities to conduct more refined forms of harassment, threats, and intimidation within virtual communities.

Summarizing, in this paper, we make the following contributions:

- We perform one of the first comprehensive security assessments of web clients of metaverse platforms from the perspective of a malicious user;
- We survey 27 metaverse platform and test three of them;
- We propose a new software-centric threat modeling approach designed to identify the security-relevant entities and attributes of metaverses;
- We propose a JavaScript global object snapshot diffing technique to identify objects related to security-relevant attributes;
- We develop and release a browser extension[1] implementing our diffing technique;
- We design and successfully execute eight attacks, including audio/video surveillance and continuous user position tracking.

2 Background

Before presenting our analysis and results, we introduce the building blocks of our paper.

2.1 Metaverse Web Client

Metaverses are virtual worlds where users interact, socialize, and play games, typically accessed through clients on platforms like MacOS, Windows, Android.These virtual environments can range from recreating public plazas [14] for social gatherings to hosting conferences where users can interact either in large groups for presentations as well as in smaller groups for private conversations [5, 9]. Recently, cross-platform web clients have emerged,

[1]The tool is available at https://anonymous.4open.science/r/big-brother-playground

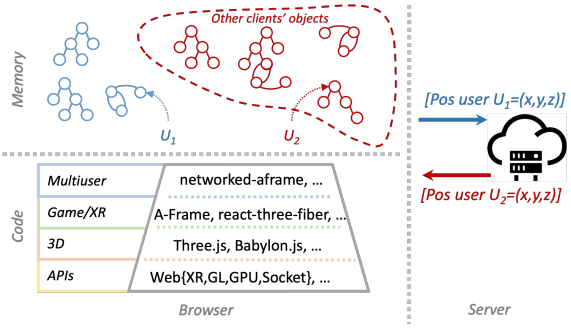


Figure 1: Components of in a metaverse web client

utilizing client-side JavaScript and libraries to create interactive, multi-user 3D environments.

Fig. 1 illustrates an example of a metaverse web client and its layered architecture. At the bottom of the stack, we find browser APIs that provide basic, low-level primitives for rendering 3D content (e.g., WebGL and WebGPU), accessing head-mounted displays and controllers (e.g., WebXR), and communicating in real-time with servers (e.g., WebSocket) and other clients (e.g., WebRTC). Above these primitives, we have JavaScript 3D engines, such as `three.js`, which offer more powerful abstractions, including cameras, lights, and materials, along with high-level operations such as creating, displaying, and animating 3D objects. On top of that, we have game and mixed reality frameworks specialized in the construction of environments, the definition of experience rules, and the interactions among various entities. At the top, we have frameworks facilitating the development of multiuser environments, providing high-level network capabilities to distribute state updates to other clients, or connecting nearby users via voice/video chat.

When the web client is executing, these libraries initialize and maintain data structures stored in the memory of the browser. These data structures usually store the parameters and the state of the 3D environment and users are updated upon a change of such state. For example, when a user interacts with the scene, e.g., it moves forward in the environment, the client pushes a state update to other clients with the aid of a server. Each client then updates the internal data structures accordingly, remaining synchronized with all other clients.

2.2 Threat Model

2.2.1 From Cheating to Security Properties. The research community has focused on the security of virtual environments due to concerns about cheating in online games, which includes the use of aimbots and wallhacks to unfairly manipulate game mechanics. However, the emergence of the metaverse, characterized by a significant social component that supports diverse interactions like virtual concerts and social gatherings, shifts this focus. In the metaverse, the emphasis on security concerns transitions from merely preventing cheating to ensuring broader security properties like authenticity, confidentiality, integrity, and availability, which are crucial for supporting its complex social dynamics.

Consider a scenario within a metaverse platform hosting a virtual conference, such as IEEE VR [5], where users can present their

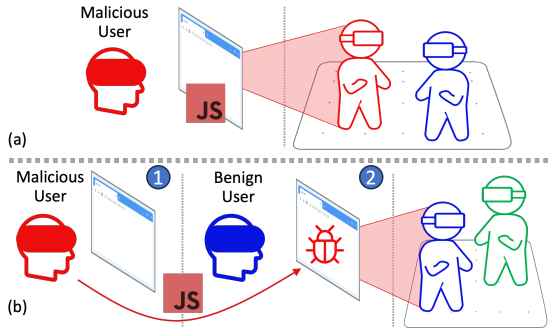


Figure 2: Attack scenarios: At the top, the attacker runs the malicious code in their browser; at the bottom, the attacker exploits a code injection to run the malicious code in the victim's browser.

work and engage in private conversations. Here, multiple groups of attendees simultaneously engage in discussions within the same virtual room, each group isolated in their own private discussion. The platform is designed to maintain privacy both audibly and visually; conversations are not audible to others beyond a certain proximity, and virtual walls render the groups invisible to anyone positioned behind them. This design prevents anyone from seeing or hearing inside these spaces unless they explicitly enter, thereby alerting the conversing parties of their presence. Such features ensure that private discussions remain confidential, emulating real-life physical constraints within the virtual environment. Therefore, the platform supports the authenticity and privacy of interactions by visibly indicating when another entity approaches their space, and by maintaining the consistency of each participant's digital identity throughout the conversation. However, if an attacker a) becomes either not visible or can ignore its visual obstructions or b) replicates an attendee's name tag and avatar to infiltrate this space, they could extract sensitive information without detection. Those breaches not only compromise privacy but also disrupt the integrity of user interactions, undermining trust in the platform's security measures.

2.2.2 Attacker Model. In this study, we focus on identifying threats that violate the security properties outlined previously and on illustrating these threats through concrete attacks against real web-based metaverses. We do not investigate classical web security threats like XSS or SQLi. Instead, we focus on how attackers can manipulate elements and properties within the metaverse, such as avatars and object positions, to violate the security properties.

We define our attacker model as a malicious user within a metaverse environment, operating independently without collusion with other malicious users or reliance on an external server to execute attacks. This attacker is a normal user, possessing no additional capabilities beyond what is typically available to any other user. The attacker begins by gathering information about the metaverse, including JavaScript code, objects stored in its memory, and network messages. This reconnaissance phase is conducted offline to study properties and state-changing operations in the platform. Once enough informations have been collected, the attacker creates malicious JavaScript scripts specifically designed for the targeted vulnerabilities. In the first scenario, as shown by the top diagram (a) in Fig. 2, the attacker enters the metaverse and directly executes these scripts from their own browser to manipulate the environment or eavesdrop on other users. In this case, the attacker can extract information directly from the browser. In a second, more covert scenario, shown at the bottom (b) of Fig. 2, the attacker uses the same initial reconnaissance and script preparation steps (1). However, instead of executing attacks directly, they deploy these scripts through a compromised system or by exploiting a code vulnerability in the web metaverse that allows JavaScript execution. This method enables the attacker to execute the script from the victim's browser (2), potentially without their knowledge. In this scenario, cross-origin requests can be used to exfiltrate data since the attacker does not have direct access to the benign user's client. The success of an attack is determined by the attacker's ability to gain more capabilites then a normal user. For example, an attack is considered successful if the attacker can overhear private conversations without being present in the same virtual space or if they obtain information beyond the visual limits set by the developers of the metaverse.

3 Methodology Overview

This section summarizes our methodology, starting from the survey of existing metaverse systems and our selection criteria of the metaverses under test (§ 3.1) and then presenting the analysis we conducted on each system (§ 3.2). Finally, this section discusses the risks of our study and presents the mitigations we put in place (§ 3.3)

3.1 Metaverse Platforms Under Test

3.1.1 Survey and Data Collection. We want to evaluate metaverse platforms that are actively used and popular. Initially, we considered using SteamVR[2] to identify relevant platforms based on a list of popular VR games (including metaverses) ranked by monthly active users. However, since SteamVR games are platform-specific, e.g., Windows, we needed a different approach for web-based metaverses.

To create our list, we first searched Google using the keywords "web" and "metaverse". We examined the first five pages of results, collecting a total of 50 web pages (10 websites per result page). We visited each website and scanned the text for platform names, then used Google Search again to find the main website for each platform.

Next, we focused on metrics to capture popularity. We first created accounts on these platforms and entered them to count the number of players in the busiest rooms. To account for different timezones, we conducted this measurement four times, spaced six hours apart. We then confirmed that each platform was utilizing the WebXR API, a browser API that enables JavaScript to interact with head-mounted displays. We also integrated data on the popularity of each platform domain name using SimilarWeb [25] and searched for the platforms on Google News to find real-world applications in events such as concerts and conferences.

[2]SteamVR is a software platform developed by Valve that extends Steam

Platforms	Visits	Scholar	Framework	Events	WebXR
Platform 1	138,800	224	A-Frame	2	Yes
Platform 2	17,000	30	Babylon.js	1	Yes
Platform 3	23,200	158	PlayCanvas	1	Yes
Dreamwave	3,900	1	*Undet.*	2	Yes
VESTA	5,000	1	three.js	-	Yes
Hyperfy	21,900	0	three.js	-	Yes
Third Room	17,000	0	Manifold	-	Yes
Ethereal Engine	11,900	0	three.js	1	Yes
VRLand	6,800	0	A-Frame	-	Yes
Ozone Metaverse	3,800	0	three.js	-	Yes
Virtual REM5	*Unrk*	0	A-Frame	-	Yes
Reach Metaverse	1,100	0	Unity	-	No
Decentraland	449,400	1520	Unity	2	No
cryptovoxels	54,900	366	*Undet.*	-	No
Spatial	657,800	29	Unity	-	No
virtway	16,000	21	*Undet.*	-	No
oncyber	65,800	11	three.js	1	No
Vircadia	32,800	9	Babylon.js	-	No
Musee Dezentral	6,000	6	three.js	-	No
Mona	39,000	2	Unity	-	No
NOWHERE	28,800	0	*Undet.*	-	No
Croquet	21,700	0	Unity	-	No
Horizon Worlds	164,500	186	*Undet.*	-	No
Arium	7,900	15	three.js	2	No
Active Replica	7,000	3	A-Frame	-	No
Portals	13,900	1	Unity	-	No
Virtual Reign	*N.a.*	0	A-Frame	-	Yes

Table 1: Survey of metaverse platforms

Finally, we completed our collection by assessing the academic relevance by counting the number of papers referring to them, using the Google Scholar search string "*platform name* metaverse".

3.1.2 Platforms for our Experiments. Tab. 1 displays the results of our survey on web-based immersive environments. Through the Google keyword search, we identified 27 immersive platforms. Of these, 12 platforms support the WebXR API and thus support head-mounted displays, enhancing the immersiveness of the experience. The remaining platforms do not support such devices, and users interact with the virtual environment through a regular display using a keyboard and mouse. As increased immersiveness may create or strengthen a false sense of privacy, we decided to focus on platforms using the WebXR API only and discarded the ones that do not.

One of the 12 platforms that supported WebXR, Virtual Reign, ceased to exist as its website is no longer accessible. We therefore discarded it from consideration. From the remaining 11 platforms, we eliminated those that lacked coverage in our Google News search, narrowing our focus to five platforms.

For our security assessment, which requires non-negligible manual work (refer to the following section), we selected three platforms only based on their usage in security, privacy, and other notable events. Platform 1 was chosen due to its use in an open source event, Platform 2 for its use in a security conference, and Platform 3 for its use in a fashion show.

3.2 Overview of our Security Assessment

A security assessment boils down to the enumeration of threats against a program. Several approaches exist to help enumerate threats [24]. For example, one can enumerate attacks starting from an attacker model and determine how such an attacker can violate a security property, i.e., attacker-centric approaches. Another approach consists of enumerating the assets an attacker may be targeting, i.e., asset-centric approaches, and determining how an attacker can violate a security property on them. The main shortcoming of both approaches is that impersonating the attacker role or the identification of assets tends to be subjective and riddled with bias, dismissing attacks because of personal opinion or beliefs [24]. As an alternative, a recommended strategy is the *software-centric threat modeling* approach [24], which is more objective, as the starting point is an over-simplified model, e.g., the Data Flow Diagram (DFD), manually curated from the software under analysis [24] (either from the code or design artifacts). The enumeration of threats is conducted for each model component, using strategies such as Per-Element-STRIDE [24]. Unfortunately, in our scenario, while the client-side JavaScript code is available, it is often minimized, compressed, or obfuscated, rendering modeling or even reversing its design diagrams hard in practice, thus hampering our ability to extract a model like a DFD.

We tackle such a challenge and propose a different approach that is software-centric at its core, but it can work in the context of web clients and, more specifically, of metaverses. We will first explore the metaverse scene to extract the entities and attributes that an attacker may want to target or exploit. We will then analyze the web client memory to find objects that implement these attributes and determine the degree of control an attacker has over them. Finally, we will determine the threats an attacker can pose by enumerating possible attacks that exploit the attributes. Specifically, our security assessment approach relies on three main steps:

Step 1: Entity-Attribute Model Extraction. We first derive a model that can identify entities and attributes of the metaverse that an attacker may want to target or exploit. We build such a model not from the code but by inspecting the metaverse as regular users enumerating the primary entities, such as the client's user, other users, objects in the scene, the scene itself, and attributes that may interest an attacker, such as users' identifiers, positions, and camera positions. For example, in a metaverse, we first identify the user as an entity. By exploring user interface interactions, such as directional keys that affect position and mouse movements that alter camera angles, we catalog attributes like avatar position and camera rotation. The goal of this step is to have a complete enumeration of the entities and attributes that populate such metaverses.

Step 2: Objects in Memory. We then analyze web client memory to find objects that implement these attributes by comparing memory snapshots after changing conditions in the environment. We label an attribute with *read* if an attacker can access its values whenever the entity's state changes. For instance, a user's position is considered readable if the attacker can access the new position from the identified object as soon as the user moves. We use the label *write* if the attacker can modify the attribute's value, and the change is propagated to other clients. For example, the user's position is considered writable if an attacker can change its state, and the change is visible to other users, indicating effective propagation across the system. This step focuses on assessing the extent of control an attacker exert over the previously identified attributes.

Step 3: Threat Enumeration and Attack Execution. The next step determines the implications of an attacker exercising control over attributes (read and write) against the security properties, i.e., *authenticity*, *integrity*, *confidentiality*, and *availability*. Identifying potential attacks mostly involves manual efforts that rely heavily on the creativity and expertise of security researchers. We start with attributes that the attacker can read, which often represent threats by themselves. For example, if an attacker can access the position of all users at any time, they can track these positions over time and infer interpersonal relationships based on users proximity patterns. We then consider writable attributes. Our analysis includes verifying if writable attributes could be used to access additional, limited attributes. For instance, if updates are received only from users within a server-side pre-defined radius, an attacker could forge rapid movements across the scene to gather position updates from more users. We also explore attacks by starting with single attributes and then combining readable and writable attributes, further amplifying an attack's potential. One such attack, which we have named *Targeted A/V Surveillance*, allows an attacker to collect real-time positions of a victim while navigating the virtual space and update only the attacker's camera (not the avatar) to the victim's position, facilitating advanced forms of *user stalking*. The goal of this last step is to determine the threats an attacker can pose by enumerating possible attacks that exploit the attributes identified in the previous step.

3.3 Ethics Considerations

This section examines potential adverse outcomes of our research and the mitigation strategies we employed.

Metaverse Platform Users The security assessments conducted in this paper might inadvertently affect other users on the platform who join the same experiences where tests are carried out. To mitigate this risk, we operated within private rooms or instances on each metaverse platform. Access to these instances was strictly regulated through invitations, ensuring entry was limited to our test accounts. This approach prevented the general public from inadvertently accessing these environments, thereby safeguarding the privacy and experience of regular users.

Before planning and executing our experiments, we confirmed that all the platforms under test offer such instances.

Platform 1 enables the creation of private, invite-only rooms. Guests can join using invite links sent by the creator, who can revoke these links after users join, preventing further access. In our analysis, we created links for our controlled accounts and then promptly revoked them to ensure exclusivity. Platform 2 allows the creation of password-protected rooms or instances, which only users with the correct password can access. Platform 3 offers to create a private instance of a room with a randomized session ID whose links were never made public.

Metaverse Platform Provider Another risk posed by our experiments is the potential negative impact on the integrity, operational stability, and availability of the platforms. Our security assessment primarily involved examining and tampering with objects stored in the web browsers' memory running on the tester's computer. As direct object modifications are confined to client-side tampering, the risk of affecting the server is minimal. Furthermore, when tampering with network update messages, our modifications did not involve the insertion of harmful payloads, such as those exploiting server-side code vulnerabilities (e.g., XSS or SQLi). Instead, our changes involved replacing a value of one type, such as a number, with another of the same type. Additionally, the volume of messages sent by our experiments was negligible compared to the regular stream of updates exchanged between clients, ensuring that our activities did not overwhelm the server with requests.

Responsible Vulnerability Disclosure Our evaluation uncovered several technical issues that could be classified as software vulnerabilities. Currently, we initiated a responsible vulnerability disclosure process with the affected parties, adhering to the following protocol. For Platform 1 we submitted the security report through the "Report a vulnerability" link on the GitHub repository and it is currently on triage. For Platform 2 , we obtained a contact point from its developer and sent them the report. For Platform 3, we obtained a valid contact point using the various communication channels offered, such as the web contact form, Twitter/X handle, and the technical support Discord channel and sent them the report. We will send reminders to the vendors, spaced two-three weeks apart, up to a period of 90 days in total. If the developers remain unresponsive, refuse to, or do not patch the vulnerabilities after this period, we will anonymize the names of the platforms in the published version of this manuscript.

4 Entity-Attribute Model

This section presents the entity-attribute model and instantiates it on our metaverse platforms under evaluation.

4.1 Construction

We conducted a detailed, systematic exploration of the core functionalities within the three metaverse platforms, deliberately excluding the interstitial menus associated with entering the virtual worlds.

Those menus, typically used for selecting which instance to enter or adjusting graphical and language settings, are not integral to the core metaverse experience. Similarly, the internal user chat is an ancillary feature like the interstitial menus. Neither element is central to the primary user interactions within the instances. Therefore, our investigation focused on the core operational features of the applications, encapsulating the main user activities available in the metaverse experience. This investigation was carried out by two researchers who extensively interacted with the platforms' user interfaces. They documented each possible user action, along with the corresponding system responses, to ensure a comprehensive mapping of actionable elements and their potential security implications within the applications.

From each action, they extrapolated the UI sequence and the subject of this action, e.g., the steps required to check where the objects are located in the room and the subject – the object coordinates –, or the steps required to change the username and the subject – the username tag. Each sequence was analyzed to establish a link between user inputs and the system's manipulation of specific data elements, discovering the underlying attributes that store it.

Following the UI analysis, the researchers conducted multi-user experiments to assess the system' physics rules. This phase aimed to a) delineate the boundaries of a regular client capabilities, such as testing the hearing range or whether teleportation was possible, and b) uncover additional attributes that were not readily accessible through the UI, e.g., the visibility of an object. These experiments provided deeper insights into the operational limits and hidden parameters of the metaverse platforms. The identified attributes were categorized into distinct application entities, and all UI state-changing operations were documented for later analysis. This structured organization streamlined the understanding of the application's architecture and improved the efficiency of subsequent security assessments.

4.2 Components

In our systematic analysis of the three metaverse platforms, we categorized the significant entities and attributes into four distinct entity types that encapsulate these virtual spaces' environmental and user dynamics. We assigned to each attribute a possible type for the object that will hold such attribute in memory. The detailed categorization of attributes, their associated entities, and the expected type are outlined in the first three columns of Tab. 2. Below, we provide a comprehensive discussion of these components.

4.2.1 Self. This entity encompasses all attributes directly associated with the client user, including:

Info: Attributes like a unique ID assigned by the system, username, friend list, and privileges defining user roles and access levels. We expect these attributes to be stored as Strings or lists of Strings.

Camera: Controls the user's visual perspective, including position, rotation, and field of view (FOV), typically anchored to the avatar but adjustable for third-person views or focusing on specific objects. Such attributes are expected to be stored in a parent camera object (as in three.js [27]) as floating-point Numbers.

Avatar: Represents the user's virtual body in the metaverse with attributes including appearance and dynamics like movement speed or body movement. The avatar movement is typically linked to camera movement. In the first-person view, the default in our selected apps is that the camera is tied to the avatar's head; in the third-person view, it hovers above the head. We expect the spatial attributes of the avatar, such as movement and position, to be Numbers or triplets of Numbers. For the Appearance, we expect complex objects with multiple children, such as geometries with materials and meshes, similar to those used in three.js [27].

Sound: Involves audio interaction capabilities of the user, such as microphone use with on/off toggles, falloff distance for sound attenuation, and modes affecting sound transmission and reception in the environment. We expect the sound to be stored as a sound object, with attributes like hearing range, falloff distance, and microphone settings stored as Numbers and Booleans.

4.2.2 Other users. Represents other users within the same environment, sharing similar attributes with the Self category but including additional functionalities that can be manipulated, e.g., visibility, an attribute often used in contexts like moderating user visibility to combat harassment. We expect the visibility attribute to be stored as a Boolean, while for the other attributes, we expect them stored as in the Self entity.

4.2.3 Objects: These are user-placed or system-generated items within the metaverse environment that possess their own set of manipulable attributes:

Info: Information like Visibility and Position that determine how and where objects appear within the scene. Similarly to previous attributes, we expect the visibility attribute to be stored as a Boolean, the position as a triplet of Numbers, and the appearance as a complex object with multiple children.

Sound: Objects can act as sound sources, contributing to the audio landscape. We expect their type to be a sound object.

4.2.4 Instance: Defines the broader environment settings that govern the collective user experience:

Properties: Such as room ID, password protection, list of users and their privileges, and settings like fly mode or privacy options (public/private). We expect those attributes to be stored as Boolean, Strings, or as list of Strings.

Scene: Static elements within the room that enhance the visual and functional complexity, such as tables or decor, each with a specific appearance. We expect the scene's Appearance to be stored as a complex object with multiple children.

5 Objects in Memory

Once an entity-attribute model of a metaverse is established, we can locate objects implementing these attributes within the client-side JavaScript execution environment stored within the browser address space. Different methods are available to access this memory.

The first method involves dumping the memory allocated by the browser process using operating system primitives, such as `gcore` in Linux systems. However, this approach is complicated by the fact that browsers are complex applications, often running multiple threads and processes and incorporating various engines like JavaScript interpreters and HTML rendering engines. These components maintain many objects in memory that do not relate to our model, which increases the complexity of our analysis.

We opted for a simpler second method, inspecting only the memory of the JavaScript interpreter. Popular browsers, including those based on Chromium and Firefox, offer developer tools for dumping the JavaScript memory. These snapshots capture all objects, including those unrelated to the metaverse. However, we observe that the objects that map to the attributes of our model are likely shared among various frameworks and components of the JavaScript program. Typically, these shared objects are properties of the `window` global object, whose inspection is more straightforward and can be performed within the same browser, e.g., through the browser console, via an extension, and the Chrome DevTools Protocol. Accordingly, we use this approach to access the memory and search for our objects of interest.

General Approach Our approach inspects a snapshot of the objects in memory, searching for the ones that are related to our attributes and finding those that can change their state. We achieve that in three steps.

Step 1: Snapshot Diffing One strategy is to cluster objects in a snapshot by type—such as strings, dictionaries, and floats—and map

Entity	Attribute	Type	Plat. 2			Plat. 1			Plat. 3		
			UI	R	W	UI	R	W	UI	R	W
Self											
Info	ID	String	–	○	–	–	○	–	–	○	–
	Username	String	●	34	B	●	556	C	–	–	–
	Friends	Strings[]	●	46(1)	–	–	–	–	–	–	–
	Privilege	String	–	○	–	–	○	–	–	–	–
Cam.	Position	3 floats(a)	●	157	A	●	71	A	●	182	A
	Rotation	3-4 floats(b)	●	194	A	●	119	A	●	201	–
	FOV	Number	–	–	–	–	–	–	–	–	–
Avatar	Position	3 floats(a)	●	288	A	●	104	A	●	174	A
	Appearance	obj, String	●	471	B	●	606	C	●	121	A
	Speed	Number	–	–	–	●	542	–	–	–	–
	Body movement	Numbers	–	–	–	–	–	–	–	–	–
Audio	Stream	Audio obj	–	–	–	–	–	–	–	–	–
	Mic On/Off	Boolean	●	153	B	●	2252	A	●	117	A
	Falloff dist.	Number	–	–	–	–	–	–	–	–	–
	Megaphone	Boolean	●	119	B	–	–	–	–	–	–
	Hearing range	Number	–	–	–	–	–	–	–	–	–
Other users											
Info	ID	String	–	○	–	–	○	–	–	○	–
	Username	String	●	9	C	●	70	–	–	–	–
	Friends	Strings[]	●	–	–	–	–	–	–	–	–
	Privilege	String	–	–	–	–	–	–	–	–	–
Camera	Position	3 floats(a)	●	39	–	●	143	C	●	226	–
	Rotation	3-4 floats(b)	●	22	–	●	325	C	●	124	–
	FOV	Number	–	–	–	–	○	–	–	–	–
Avatar	Visible	Boolean	–	–	–	–	○	–	–	–	–
	Position	3 floats(a)	●	39	–	●	112	C	●	226	C
	Appearance	obj, String	●	367	C	●	1951	C	●	34	C
	Speed	Number	–	–	–	●	201	–	–	–	–
	Body movement	Numbers	–	–	–	–	–	–	–	–	–
Audio	Stream	Audio obj	–	–	–	●	–	–	–	–	–
	Mic On/Off	Boolean	●	10	–	●	59	–	●	97	–
	Falloff dist.	Number	–	–	–	–	–	–	–	–	–
	Megaphone	Boolean	●	9	–	–	–	–	–	–	–
	Hearing range	Number	–	–	–	–	–	–	–	–	–
Objects											
Info	Position	3 floats(a)	●	15	–	●	319	C	–	–	–
	Appearance	obj, String	●	64	–	–	–	–	–	–	–
	Visible	Boolean	–	–	–	–	○	–	–	–	–
Audio	Source	Audio obj	–	–	–	–	–	–	–	–	–
Instance											
Properties	ID	String	–	○	–	–	○	–	–	○	–
	Password	String	●	–	–	–	–	–	–	–	–
	Admins	Strings[]	●	304*	–	–	–	–	–	–	–
	Members	Strings[]	●	212*	–	●	399	–	–	–	–
	Fly mode	Boolean	●	119*	–	–	–	–	–	–	–
	Privacy	String	–	–	–	–	–	–	–	–	–
Scene	Appearance	obj, String	●	10436	–	●	7562	C	–	–	–

Table 2: The entity-attribute model istantiated on the three metaverses under test. *: Only when the user (self) is an admin; ○: When the the attribute was found via manual analysis; (a): <x,y,z>; (b) <x,y,z> or <x,y,z,w>; (1): The property is not accessible anymore in the latest version of the Metaverse; A: Direct Object Editing, B: State Update Functions, C: Network Update Functions.

these clusters to attributes using the expected type column (see Tab. 2). However, this approach may fail as many objects share types, resulting in large clusters that require manual analysis, with no clear signal that an object is linked to an attribute.

Another strategy involves comparing snapshots taken at different times, with the second snapshot occurring *after the tester changed a condition in the environment related to the attribute of interest*. For example, assume the tester seeks to identify attributes related to their own presence, i.e., the "self" entity. They take one memory snapshot right after the environment loads and a second after moving their avatar a few steps forward. By comparing these snapshots, the tester can identify objects that remained unchanged after the movement and discard them as unrelated to the controlled variable. The remaining objects are either related to the movement or altered by coincidence. Repeating this differential analysis with additional snapshots can reduce the number of objects to inspect to a manageable amount. We detail this technique in § 5.1.

Step 2: Mapping Objects to Attributes Following this initial filtering, the tester refines the analysis by categorizing the remaining objects according to expected types and values, stopping at the first confirmed match for each attribute. For specific, known values like usernames set during testing, the tester identifies relevant objects that match the given value. For attributes with unknown values, the tester relies on the expected types predetermined in our threat modeling to trace objects associated with the attributes. We detail this step in § 5.2.

Step 3: Writable Objects In the final stage, the tester tested the writability of these objects to determine if modifications could impact the metaverse's state and propagate to other clients. This involved tampering techniques such as directly editing object properties, using internal framework functions to trigger updates, and replaying modified network messages. We detail this step in § 5.3.

5.1 Memory Snapshot Diffing

This section presents the algorithms to create snapshots and how we calculate differences, as in Algorithm 1.

Algorithm 1 Memory Snapshot Diffing.

```
1:  S := CreateSnapshot(window)
2:  c_N := 0
3:  repeat
4:      wait(TIME)
5:      S := RemoveChangedObjects(S, window)
6:      c_N += 1
7:  until c_N < N
8:  c_T := 0
9:  repeat
10:     TesterAction()
11:     S := RemoveUnchangedObjects(S, window)
12:     c_T += 1
13: until c_T < T
```

5.1.1 Create Snapshot. The algorithm starts by taking a snapshot of `window` recursively and records each object's path and content. The path is a dot-concatenated string of property names and array positions. For example, the path of the object pointed by the property `window.obj1.obj2[1].obj3` will be `obj1.obj2.1.obj3`. The

algorithm starts from the `window` object and enumerates the properties. From these properties, it discards those that are redundant ones. Redundant properties such as child, parent, and siblings, which are merely shortcuts to other accessible objects, are discarded. For each property, the algorithm checks whether it is of a primitive type (e.g., Boolean, Number, and String). If it is, a copy of the value and its path is saved. If the object has a non-primitive type, it checks whether we have visited it before to avoid loops. Unvisited objects are processed recursively. Instead of a simple recursive call, the algorithm conducts checks to reduce the number of objects saved and minimize iteration cycles. Unvisited objects are tested for serializability via the `structuredClone()` function. Serializable objects, which do not contain loops, are then considered. These may hold other relevant objects but are not immediately extracted; instead, following a lazy approach, they are marked for later examination during snapshot comparisons. If found serializable, the object is cloned and stored along with its path. Recursive examination is limited to non-serializable objects.

5.1.2 Diffing Snapshots. The general idea is to identify those objects that have changed and those that have not *when the tester changed a condition in the environment*. However, such an approach typically requires taking multiple snapshots of the `window` object (before and after the change) and searching for altered objects, which is resource-intensive.

To optimize this process, we introduce a more efficient approach. First, as many objects change immediately after the first snapshot and before the tester changes the condition, it is possible to identify and discard these objects, thereby narrowing down the number of objects under consideration. Second, taking multiple snapshots to identify changes is not always necessary. Given that our snapshot retains exact copies of the objects, we can directly compare the objects in the current `window` with the ones in the snapshot. Moreover, since we have stored the path of each object, there is no need to search for similar objects. Instead, we can directly access each object in `window` using the path stored in the snapshot.

Accordingly, we define and use two types of diffs. The first diff occurs after the initial snapshot but before the tester modifies the environment condition. This diff removes from the snapshot all objects that have changed in the `window` object. The second diff happens after the tester has changed the condition, e.g., moved the avatar or added another user. This diff removes any objects that have not changed from the remaining objects in the snapshot.

1. Remove Changed Objects This function iterates over the paths in the snapshot. For each path, it retrieves the corresponding object from the global object. If the object no longer exists at its previously recorded location, it is either deleted or relocated. Such objects are marked as changed and removed from the snapshot. If the object still exists, it is compared with its snapshot counterpart. This comparison is conducted using type equality for primitive types or via a string representation (using the `JSON.stringify()` method) for objects, making the comparison more efficient. If the comparison shows differences in primitive types, the object is removed from the snapshot. If they differ in the string representation, it indicates that either a nested object has altered or the entire object has changed. We then recursively explore both objects to identify identical nested objects, which are then retained in the snapshot, while all altered nested objects are discarded. Finally, if the `JSON.stringify()` comparison fails, it indicates that the object now includes a loop. Therefore, the object is visited recursively to identify and retain nested objects that remained unchanged, adding them to the snapshot.

2. Remove Unchanged Objects This function is similar to the previous one. However, instead of removing changed objects, it removes unchanged ones. For each path in the snapshot, it checks for the string representation of each object in the `window` with the same method as before. If the value within the current object has changed, that object is marked for removal. Conversely, if the value remains unchanged, the object in the snapshot is updated with the new value. If retrieving the value for a given path fails, it indicates that the object has been deleted or moved. In such cases, the value at the observed path is considered changed, and no further action is needed for that object.

5.2 Mapping Objects to Attributes

We now demonstrate how a tester can utilize the memory snapshot diffing algorithm (Algorithm 1) to identify objects associated with an entity's attributes.

5.2.1 Tester Actions. The tester employs Algorithm 1 for each entity and attribute, adjusting only the manual components, i.e., step 10. The other steps are fully automated functions given by the tool. Depending on the entity involved the tester will control one or more user accounts. For instance, to analyze attributes related to position, the manual action involves walking inside the scene. For the *Self* entity, the tester uses only one account—the attacker's—to manually move the avatar and use the tool function for the diffing algorithm. For the *Other users* entity, the tester operates two accounts: one for a user who performs the movement and another for the attacker, who does not move but executes the diffing algorithm after the movement action. The tester executes Algorithm 1 for each attribute to obtain a corresponding list of objects, which may still include unrelated objects. To refine this list, parameters `N` and `T` can be tuned to control the object removal iterations in steps 3-7 and 9-13 of Algorithm 1, respectively. The wait time between removals, `TIME`, is adjustable in step 4. Our empirical settings of `N=1`, `TIME=10s`, and `T=3` optimize both the speed and thoroughness of the reduction process.

5.2.2 Values Analysis and Attributes Types. Finally, the tester engages in a filtering process to review the remaining objects, further refining the selection. This filtering isolates only those objects that accurately represent the attribute. Depending on the degree of control the tester has over the attribute's value, two distinct filtering approaches can be employed:

Chosen Values When possible, the tester preselected the attributes values during account setup or while preparing the memory snapshot diffing algorithm. For instance, usernames were set to unique strings, which were then located within objects containing user data or scene elements, such as the nametags displayed above avatars. The tester searched for these strings within the objects.

Expected Types For cases where the values after the action were not predetermined, the tester depended on the expected types

identified during the modeling of the metaverse platforms, as documented in § 4. This method enabled the tester to locate objects tied to user position, rotation, and movement speed by identifying triplets of numbers that responded to movements. Similarly, for attributes like appearance, the tester searched for string patterns corresponding to URLs or paths of 3D models and specific structural or naming conventions used by the 3D libraries, such as UUIDs or meshes for three.js. The tester then followed the hierarchy of parent objects to map the entire appearance structure.

5.3 Writable Objects

After identifying and mapping objects to attributes, the tester verified if a change to the object results in a permanent state change in the metaverse platform. Although this step was predominantly manual, the tester followed a structured analysis by employing three main techniques to trigger a change of state in the metaverse:

A. Direct Object Editing. In this case, the tester locates the object using the path from the snapshot and modifies the object's properties consistently with the attribute's meaning and data type. The tester then verifies whether the change was reflected in the scene. For example, if the tester updates the position of another user, the tester monitors from both the user's account and the tester's account if the position of the user has changed to the new location. If so, the attribute is labeled as *writable*. It may happen that the new value in the object is replaced by the original one. In these cases, there may be a component in the code with direct access to the same object that keeps on rewriting our changes. In these cases, we attempt to break that reference by cloning the object, modifying the clone, replacing the original with the clone, and observing the expected change in the rendered environment.

B. State Update Functions. Even if a direct edit permanently changes the value in an object, it may not result in a state change in the metaverse. This may occur because other objects also need to be edited consistently, suggesting a function that could trigger the state update across multiple objects. Given that the code can be obfuscated, finding these functions may not be straightforward. Instead of searching for the function in the source code, we attempt to find it at runtime the moment the object is accessed. Accordingly, the tester assigns a `getter` and a `setter` callback function and sets a debugger breakpoint at the first instruction of our callback function. Whenever the update function accesses the object, the JavaScript engine interrupts the execution at the breakpoint. The inspection of the call stack will reveal the functions accessing the monitored object and its parameters. The tester saves the function names and parameters as global objects and forcefully executes the functions and observes the expected changes.

C. Network Update Functions. Forced execution of functions may not always be effective. The program could be in a state that causes the function to fail due to some uncontrolled state object. In these instances, the tester monitors the network traffic, particularly focusing on WebSocket communications—a technology that enables interactive, real-time data exchanges between the client and server. The tester looks for messages within this WebSocket traffic that might contain strings related to the intended update. If such a message is found, the tester forges one with a value of their choice and observes if the change is propagated to other clients.

5.4 Results

5.4.1 Readable Objects. Our application of Algorithm 1 and the following analysis provided concrete insights into the management of sensitive attributes. Below, we summarize the results obtained from three different applications. The full results are detailed in the *R* (Read) columns of Tab. 2.

Platform 1 Results For Platform 1, 17 out of 18 editable attributes were successfully detected. The search began with the `window` object, which initially contained over 500k objects. The results of the filtering process can be seen in the read (R) column of Tab. 2. Filtering was done primarily utilizing Expected Types, with the Chosen Value heuristic applied to the usernames. The time required to finish the process was 8-10 hours for the tester.

Platform 2 Results In Platform 2, our analysis identified 21 out of 23 modifiable attributes through the user interface using our tool. The analysis began with the `window` object, initially containing over 500k objects. The filtering process effectively reduced this to a manageable number for each property, see read (R) column in Tab. 2, highlighting the tool's ability to isolate relevant attributes. The primary method for filtering used Expected Types, while the Chosen Value heuristic was applied to the usernames. The time required to finish the process was 8-10 hours for the tester.

Platform 3 Results For Platform 3, ten attributes modifiable via the user interface were identified. The analysis started with the `window` object, initially containing over one million objects. The results prior to filtering can be seen in the Platform 3 read (R) column of Tab. 2. The filtering process exclusively utilized Expected Types, as no user-chosen values are applicable to the properties analyzed. Due to the absence of certain attributes and entities found in other platforms, the time required to analyze Platform 3 was approximately 4 hours.

Despite the significant reduction in the number of objects in all applications, the large average number remained primarily due to the complex nature of 3D objects, specifically those related to the Appearance attributes. These objects typically consist of multiple components, such as polygons, shaders, textures, and materials, contributing to the high count of objects detected by the tool.

5.4.2 Additional Objects. During the manual filtering process, we identified objects in memory corresponding to attributes from our threat model that were not modifiable through the UI. Examples of these objects include user and instance ID strings, the user camera's FOV, and the visibility flags for other users. We indicated these objects in the *R* column of Tab. 2 with a hollow circle.

5.4.3 Writable Objects.

Platform 1 Our tests in Platform 1 enabled us to evaluate the writability of 12 attributes. Here, we primarily utilized the technique of replaying altered network messages. By replicating and modifying network calls, we were able to confirm changes of state, providing insights into the data synchronization process within client-server architectures. The analysis duration was 12 to 15 hours.

Platform 2 In this virtual environment, we successfully assessed the writability of nine specific attributes. Using our algorithms, we directly modified JavaScript objects for spatial properties such as Position and Rotation, with changes immediately visible in the application. For attributes related to user settings and identities, like Username and Appearance, more complex methods, including function interception and call stack analysis, proved effective. The analysis took between 12 to 15 hours.

Platform 3 In Platform 3, we identified six writable attributes. We had success through direct edits to object properties and network manipulation. These methods allowed us to perform state-changing operations, in some cases visible to all users except the designated attacker or victim. For example, altering the position of a player do not affect the victim's client but is observable by all others. The analysis was completed in 8 to 10 hours.

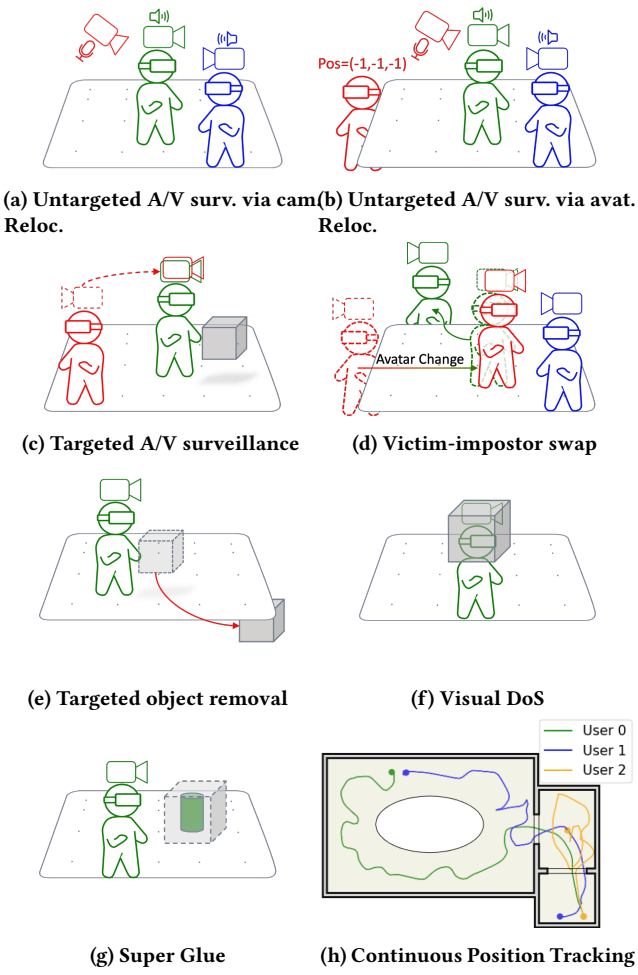
6 Attacks

The analysis in § 5 returned a list of attributes and the type of access allowed to an attacker, i.e., *read* and *write*. In particular, for writable attributes, § 5 identified how an attacker could modify that attribute, e.g., via direct object modification, function updates, or network messages. These results constitute the building blocks to identify attacks, which we cover in this section.

Brainstorming Attack Possibilities Finding attacks given a set of permissible object change operations is not trivial, and it is fundamentally a creative and manual activity that deeply relies on the expertise of the security researcher. In this paper, a team of two researchers enumerated possible attacks in multiple brainstorming sessions, setting as a goal the violation of one of the four security properties, i.e., *authenticity*, *integrity*, *confidentiality*, and *availability*, assuming the attacker can read and write the attributes listed in Tab. 2. When one researcher verified that an attribute can be tampered with an arbitrary value, they explored the attack potential by combining attribute operations. After enumerating possible attacks, the researchers (i) implemented the attacks against the platforms, (ii) set up the scene with the entities, e.g., victims and objects, (iii) determined if the attack achieves the intended goals, (iv) determined if the attack has side effects that could hinder its effectiveness or impact, and (v) evaluate exploitability under our thread model (e.g., threat model 2a or 2b).

Overview of the Results In total, we identified 10 attacks, out of which we successfully implemented eight, ranging from audio/video surveillance attacks, continuous user position tracking, impersonation attacks, and experience tampering attacks. For each attack and platform, we developed a JavaScript script that automates the entire attack process. Four of these attacks were executed against Platform 2 and Platform 3 and all eight against Platform 1. The attacks are summarized in Tab. 3.

Attack Descriptions Template We follow a strict template for each attack. First, we summarize the attack. Then, we technically explain how the attack is performed and show the involved attributes. Finally, we discuss risks and impact of the attack and discuss their exploitability.



(a) Untargeted A/V surv. via cam Reloc.
(b) Untargeted A/V surv. via avat. Reloc.
(c) Targeted A/V surveillance
(d) Victim-impostor swap
(e) Targeted object removal
(f) Visual DoS
(g) Super Glue
(h) Continuous Position Tracking

Figure 3: Attack scenarios for an active attacker. The attacker is in red. The victim is in green. In blue is a user in the social circle of the victim.

6.1 Audio/Video Surveillance

The first attack allows an attacker to implement audio/video surveillance on users in a metaverse. We distinguish two different variants of attacks, exploiting different attributes. The first variant is untargeted, where the attacker places the camera and microphone in an area of interest. The second variant is targeted, where the attacker places the camera in the same position as the camera of the targeted victim.

6.1.1 Untargeted A/V Surveillance via Camera Relocation. This attack allows an attacker to covertly eavesdrop on conversations and have a video feed of an area of a virtual environment. It achieves that by moving the camera position to a selected area. As the position of the camera and the avatar are not linked to one another, the attacker's avatar can remain in another location (Fig. 3a).

Technical Explanation This attack leverages the decoupling of the camera, which carries the audio source, from the avatar's position. The attacker either a) edit directly the objects values or

	Attack			Platform			Self										Other										Objects		Instance			
Attack name	Attack Scenario	Active	Passive	Plat. 2	Plat. 1	Plat. 3	ID	Name	Friends	Pos	Rot	Pos	Rot	App	Mic	Megaphone	ID	Name	Pos	Rot	Pos	Rot	App	Mic	Megaphone	Position	App	Admins	Members	Fly	Assets
Untargeted A/V Surv. – Camera (Fig. 3a)	a	●		●	●	●				W	W																				
Untargeted A/V Surv. – Avatar (Fig. 3b)	a	●		●	●	●						W																			
Targeted A/V Surveillance (Fig. 3c)	a	●		●	●	●				W	W						R		R	R											
Continuous Position Tracking (Fig. 3h)	a,b		●	●	●	●											R	R			R								R		
Victim-Impostor Swap (Fig. 3d)	a	●		○	●	○		W						W			R	R			W										
Targeted Object Removal (Fig. 3e)	a,b	●		○	●	N/A											R									W					
Visual Obstructions (Fig. 3f)	a,b	●		○	●	N/A											R				R					W					
Super Glue (Fig. 3g)	a,b	●		○	●	N/A											R									R/W					

Table 3: List of attacks, type of attacker (active/passive), the attack scenario, the affected platform, and the attributes (read R and write W).

b) manipulates network traffic to freeze their avatar in its current position by capturing and resending its last known coordinates through WebSocket traffic, making it appear stationary to other users. Then, the attacker freely places their camera in a fixed area of interest, enabling listening to the various conversations without visual indication of their presence. This separation of audio and visual presence exploits the writable attributes of the camera's position and rotation, allowing for surveillance tools.

Risks and Impact We implemented and verified the effectiveness of this attack in all three platforms. The main risk is unauthorized access to private conversations and interactions, breaching user privacy and potentially gathering sensitive information without consent. This could be particularly damaging in environments where users assume a level of privacy and confidentiality.

Exploitability This attack manipulates the objects identified through our methodology, resulting in the display of the camera view and the relaying of audio to the browser used to execute the attack. An attacker can execute this attack under the scenario shown in Fig. 2.(a). However, when the same manipulations are executed by the benign user of the attack scenario Fig. 2.(b), their user experience will be disrupted, making these manipulations insufficient to carry out the attack. Although we were unable to find a non-disruptive alternative, we cannot rule out the possibility that such a variant exists.

6.1.2 Untargeted A/V Surveillance via Attacker's Avatar Relocation. This attack achieves the same result as the previous one. However, this attack exploits the writable avatar position attributes in virtual environments, allowing an attacker to render their avatar invisible by positioning it outside normal interaction boundaries. By manipulating the avatar's position attributes, the attacker can effectively make their avatar disappear or become hidden from other users. This is achieved by moving the avatar outside the spatial boundaries of the metaverse map, such as outside a room's walls or above the ceiling, thereby making it invisible to other users (Fig. 3b).

Technical Explanation The attack involves accessing the avatar's position attribute within the virtual reality environment's client. The attacker identifies the avatar's current position and then modifies these coordinates to values that position the avatar outside the usual boundaries of the map. This manipulation is achieved by adding offsets to the current position, ensuring the avatar moves to a location where it cannot be seen by others.

Risks and Impact We implemented and verified the effectiveness of this attack in all three platforms. The capability to render avatars invisible presents significant risks on platforms like Platform 2 and Platform 1, where avatar position attributes are directly modifiable. Key risks include the same as the previous attack. In addition, hiding the avatar includes evasion of user observation, which can facilitate unauthorized and harmful activities unseen by others. This vulnerability can serve as a foundation for covert operations within these virtual spaces, allowing attackers to conduct passive attacks that can result in surveillance.

Exploitability As with the previous attack, the manipulations in this attack disrupt the user who is unknowingly running them (see the attack scenario in Fig. 2.(b)). The current implementation of this attack is practical only in the scenario depicted in Fig. 2.(a).

6.1.3 Targeted A/V Surveillance. This is similar to the untargeted A/V surveillance attack. However, the attacker now positions their camera in the same position as the victim's. The attacker monitors the victim's position and updates the cameras accordingly in real-time. Such an attack allows the attacker to hijack the point of view of another user's avatar, seeing through their perspective without moving their own avatar (Fig. 3c).

Technical Explanation In Platform 2 and Platform 1, this attack is executed by manipulating WebSocket traffic to separate the control of the attacker's camera from their avatar and align it with the target's camera viewpoint. This involves key attributes such as the attacker's camera position and rotation, along with the target's ID, position, and rotation. Initially, the attack halts updates of its movements to other clients to prevent the avatar from mirroring camera movements. Concurrently, the attacker's camera position and rotation are programmatically adjusted in real-time to synchronize with the target's camera in each frame. This adjustment mirrors the target's view precisely, creating the illusion that the attacker is seeing from the target's perspective. In Platform 3, the approach differs slightly due to its camera mechanics, which always point towards the user. Here, both the camera position and user position objects are directly edited to ensure they always align with the victim's rotation. By positioning the attacker's avatar far away

from the victim, it remains hidden from the victim's view. However, the attacker can see from the victim's perspective, effectively hijacking their viewpoint.

Risks and Impact We conducted this attack and verified its severity on all three platforms. Such an attack poses severe privacy risks by allowing unauthorized access to a user's visual and spatial perspective. Additionally, when compared to the untargeted version, this variant is significantly more severe as it could pave the way for new and more pervasive safety concerns perpetrated by abusive partners and stalkers against vulnerable user groups, aiming to harass, scare, and threaten them.

Exploitability As with other attacks in this category, this attack currently manipulates the client state, rendering it applicable primarily under the scenario depicted in Fig. 2.(a).

6.2 Continuous Users Position Tracking

On top of A/V surveillance, an attacker can also track the real-time positions of all users in a metaverse room. By recording these positions along with timestamps, the data can later be used to plot interactions and create a heatmap of connections between players.

Technical Explanation The data collection is executed by initiating a position-tracking function that captures the positions and usernames of all players in the environment at regular intervals. This function accesses and iterates over player entities, storing each player's username and position at regular intervals. The collected data can be processed both offline and online (while the attack is happening). Fig. 3h illustrates the position-tracking activity that an attacker can perform against three users inside a virtual space.

Risks and Impact We executed this attack in all platforms. The primary risk of this attack is the potential breach of privacy by exposing user movement patterns without their consent. The data gathered can be used to infer user behaviors, interactions, social connections, and habits within virtual spaces.

Exploitability This attack operates without requiring a client interface to display outputs, making it covert and compatible with both attack scenarios. In the case of a remote attacker, as depicted in Fig. 2.(b), cross-origin requests can be utilized to exfiltrate data.

6.3 Impersonation Attacks

6.3.1 Victim-Impostor Swap. This attack involves an attacker impersonating another user by 1) changing their own avatar's name and appearance to that of the target, 2) teleporting the target user to an out-of-bounds location, and 3) moving the attacker to the original victim's position. The final goal of this attack is to replace a victim with the attacker without others noticing (Fig. 3d).

Technical Explanation The attack manipulates WebSocket communications to alter the attacker's username and avatar to resemble another user's while concurrently relocating the other user. The attack process begins with the attacker obtaining the unique ID of another player. Following this, the attacker overrides WebSocket functions that are responsible for message handling. This interception facilitates key manipulations: 1) extracting critical information from the victim such as their position, rotation, appearance, and nametag by matching messages containing victim's ID; 2) altering the attacker's own username and avatar to exactly match that of the target; 3) teleporting the victim to an out-of-bounds location to isolate them from the main interaction space; 4) the attacker relocates to where the victim was originally positioned, thereby assuming the victim's place in the virtual environment. This sequence of actions enables the attacker to seamlessly take over the victim's identity without alerting other users.

Risks and Impact We conducted this attack on Platform 1. However, we could not reproduce it on Platform 2 because we could not find a way to write the other users' position attributes. On Platform 3, while we can alter other users position, usernames are created at sign up and cannot be changed, so the attack is not possible. A successful attack results in user impersonation and dislocation, potentially facilitating an attacker's use of social engineering techniques.

Exploitability Similarly to the A/V surveillance attacks, this attack alters the client state. Therefore, this attack is currently possible under the scenario illustrated in Fig. 2.(a).

6.4 Metaverse Experience Tampering

Some Metaverses can allow users to add objects to the scene. For example, Platform 1 offers catalogs of objects that users can browse and add to the scene or load via URLs. These objects play an important role in the various experiences. For example, they can be used for decorating a room as well as key elements in the logic of a game, such as marking a path already explored in a maze immersive game, e.g., the Maze Challenge of Platform 1. Other platforms, like Platform 3, do not offer object spawning capabilities, making this category of attacks not feasible. In this section, we present attacks that leverage these objects (Fig. 3e – Fig. 3g).

6.4.1 Targeted Objects Removal. The first attack targets the objects a specific user places in the scene. The attacker continuously detects, and teleports the objects a specific player creates to an out-of-bounds location as soon as they are spawned (Fig. 3e).

Technical Explanation The attack is executed by setting up WebSocket monitoring to detect object creation events. A custom function intercepts WebSocket messages that indicate the creation of new objects by the targeted player. This involves overriding the WebSocket's functions to manipulate data. The steps include detecting the creation event of an object by the target player, extracting the object's unique ID, and constructing a new WebSocket message that changes the object's position to teleport it out of bounds. This method relies on identifying the user and object creator by their IDs and manipulating the location of the object.

Risks and Impact We tested this attack against Platform 1. Such an attacker can severely disrupt the gameplay or interactive experiences of targeted users by rendering their ability to place and use objects in the environment ineffective. This disrupts the player's ability to interact meaningfully within the environment, resulting in a denial of service for the user's virtual objects.

Exploitability Our analysis confirms that this attack is viable under both attack scenarios explained in Fig. 2 since it does not affect the client that executes it. The script that removes objects from the scene can be launched directly from an attacker's client or executed covertly via a benign user's client.

6.4.2 Visual Obstructions. This attack exploits the functionality within virtual environments that allows dynamic control over object

positioning relative to user viewpoints. An attacker can deploy one or multiple objects, read the position of a victim's camera, and use a function to continuously update the object's position to coincide with the camera's view (Fig. 3f). This results in rendering the player "blind" by obstructing their visual field entirely, thus significantly degrading the user experience. Not only does this impair the victim's ability to navigate and interact within the virtual environment, but it also makes their avatar unrecognizable to other players, further isolating the victim.

Technical Explanation The attack uses the manipulation of object positioning mechanics in virtual environments, positioning at the victim's camera movements. The attacker uses a function that automatically spawns objects directly in front of the victim's camera. This script continually queries the current position of the victim's camera and adjusts the coordinates of the objects, ensuring they always remain in the direct line of sight.

Risks and Impact We implemented and verified the effectiveness of this attack in Platform 1. The primary risk of this attack is the degradation of user experience through visual impairment, which can lead to disorientation and degradation of the experience. Additionally, by obscuring the victim's avatar, the attack can isolate the victim from social interactions within the virtual space, effectively rendering them unrecognizable to other users. This not only disrupts individual gameplay or participation but also poses broader implications for social dynamics and safety in virtual environments.

Exploitability This attack can be executed under both attack scenarios of Fig. 2. The script that obstructs a victim's visibility can be launched directly from an attacker's client or executed covertly via a benign user's client.

6.4.3 Super Glue. In this attack, the attacker places transparent objects in the scene for each object that a user places in the scene. The attacker then continuously updates the position of the transparent object to match the position of the object placed by the user. This results in the user being unable to move the object, as the transparent object will always be superimposed (Fig. 3g).

Technical Explanation A function continuously checks the objects in the scene and if it detects a new object placed by the user, it spawns a transparent object at the same position. The attacker then uses a loop function to update the position of the transparent object to always match the position of the object placed by the user. This results in the user being unable to move the object, as it will return to the initial position (on top of the user object) as soon as the user tries to move it.

Risks and Impact We implemented and verified the effectiveness of this attack in Platform 1. This attack can hinder the user's ability to interact with the environment by rendering objects immovable. It affects only the designated user, not the client executing the attack, and can be carried out from either the attacker's or a benign user's client, as shown in Fig. 2.

6.5 Partially Feasible Attacks

Various attacks were conceived and tested, yet they failed to function as intended. This section presents our attempts, the expected impacts, and the technical explanation that prevented their success.

6.5.1 Teleporting Players to Arbitrary Location. In this attack, we attempted to manipulate the position of a target player by modifying the WebSocket message that contains the player's coordinates. The modification included the target's ID and the desired coordinates for teleportation. We teleported the victim to a specified location; however, teleporting the victim froze their character. The only option for the victim is to rejoin the metaverse room. Freezing arose because the platform reassigned the owner ID of the victim's avatar to that of the attacker's ID during the teleportation process. Consequently, the victim's client no longer recognized the avatar as its own, blocking any movement commands from the victim.

While freezing is acceptable for attacks like the Victim-Impostor Swap, it prevent exploring the full attack potential of teleporting, including causing motion sickness by injecting visual vibrations or hijacking users' trajectories (such as in the human joystick [4]).

6.5.2 Controlled Camera Movement. This attack aimed to manipulate the camera's position or rotation directly, independent of the avatar. This manipulation could induce disorientation or dizziness in the victim or facilitate social engineering attacks by deceiving the victim about their actual environment. For example, it makes users believe they are acting on some object in a specific room while they are interacting with another object elsewhere. However, we found that the camera's position and rotation are tied to those of the avatar. Thus, altering the camera position without simultaneously changing the avatar's position or orientation proved ineffective.

6.5.3 Reasons for Failure. The failure of certain attacks in our study is primarily attributed not to robust client-side security measures, but to specific limitations imposed by the server-side architecture or the absence of certain features in the virtual environments.

Missing Features In environments like Platform 3, the absence of user-editable objects directly prevents the execution of object-related attacks (Fig. 3e – Fig. 3g). This is indicated as N/A (not applicable) in Tab. 3, as the platform does not support features necessary for these attacks.

Property Modification Failures Some properties were not successfully altered in the state. For instance, in Platform 2, attempts to alter avatar or object positions through network updates were not propagated by the server. Without access to the server-side code, it is unclear whether this results from intentional security measures or merely design side-effects. The non editability of a) other users avatar and b) object location stopped us to execute some attacks (Fig. 3d – Fig. 3g) on Platform 2.

7 Discussion

7.1 Results of our Assessment

In this section, we discuss a few aspects and implications of our security assessment, focusing on the severity of attacks, vulnerabilities, comparisons with multiplayer gaming platforms, and specific design choices made by Meta.

Severity of Attacks and New Possible Malicious Actors. Our research identifies not only individual-targeted attacks but broader, severe threats affecting user privacy, availability, and the integrity of the environment. In social-centric virtual spaces like metaverses, such

vulnerabilities could be exploited to stalk or abuse users. An attacker can concretely use the scripts in § 6 in two scenarios. First, a malicious user runs them in their own browser while being in the environment. Second, the attacker does not join the environment but manages to execute arbitrary JS on the client of a benign user (e.g., via XSS vulnerability). As metaverses gain traction virtual offices or business meetings, they may attract more sophisticated attackers, including organizations engaging in corporate espionage or state-sponsored entities gathering intelligence. This potential for diverse misuse underlines the critical need for robust security measures tailored to the evolving landscape of virtual environments.

Vulnerabilities in Metaverse Platforms. The complexity of security issues in metaverses necessitates a multifaceted mitigation approach, including code patches and fundamental redesigns of critical components. From our security assessment, we can group vulnerabilities into two violations of secure engineering principles. First, the *need-to-know principle* is often breached [12, 31], with clients accessing excessive information, undermining the system's confidentiality (i.e., continuous users' position tracking). Second, many platforms lack or poorly perform server-side validation of global state updates [2, 8], allowing attackers to manipulate the environment.

Adapting Attack Models for Web Metaverses. Attacks like those we proposed are also present in gaming, even if with different objectives. Web applications, however, continuously fight vulnerabilities that allow the execution of malicious JavaScript code, such as XSS, script gadgets, and DOM clobbering. This battle against web-specific threats is far from over and remains an ongoing challenge. Consequently, the potential severity of these attacks could be greater on web-based platforms, where attackers might also exploit these code execution vulnerabilities (refer to the bottom schematic in Fig. 3) to execute attacks more covertly.

Comparison with Multiplayer Gaming Platforms. The gaming industry's reliance on client hardening, such as restricting access to memory and network stacks through anti-cheat technologies [18], offers an interesting parallel to metaverse platforms. Anti-cheat techniques have historically initiated a cat-and-mouse game, where, eventually, game cheaters have the upper hand. Additional countermeasures exist, such as obfuscation, hiding data structures, and memory integrity checks. However, they all serve as temporary deterrents against attacks and are not long-term solutions.

Recently, the use of secure enclave technologies, e.g., SGX [1, 10], has shown potential in mitigating data tampering, suggesting a promising avenue for enhancing security in metaverse environments. Future research should investigate how such technologies could be adapted from gaming and the web to virtual worlds, especially given the increasing interest from strong adversaries.

Transferring Existing Defenses to the Web Platform. Transferring gaming anti-cheat measures to web-based metaverse platforms is challenging due to the limitations of web browsers. Unlike gaming systems, where anti-cheat mechanisms are integrated deeply with the operating system to monitor and restrict memory access, browsers lack such in-depth system control. Such anti-cheat systems would safeguard against unauthorized access and manipulation of data, thus preventing malicious interference. However, while securing JavaScript execution could enhance defense, it requires careful consideration of potential trade-offs. Hardening JavaScript execution to prevent tampering could make malicious scripts harder to detect, monitor, and mitigate, enabling more covert and potentially dangerous attacks. Therefore, the trade-off between securely growing these platforms and potentially worsening security problems in the web ecosystem must be carefully considered.

Transferring Client-side Logic to the Server. Transferring all security responsibilities to the cloud by processing graphics on the server side, akin to game streaming platforms, represents another potential strategy. Meta, as we discoverd by analyzing their platform, has already adopted this method in their web metaverse [30], shifting all client-side logic to the server and using the browser merely to stream video and gather user inputs. This approach addresses security concerns by centralizing control but introduces challenges, such as the need for low-latency network connections to prevent user experience degradation. While effective for a resource-rich organization like Meta, it remains impractical for smaller entities, highlighting a divide in potential security strategies based on organizational resources. This disparity raises questions about such models' scalability and applicability across different sizes and types of organizations operating within the metaverse.

7.2 Limitations

Threat Modeling and Testing. Our threat modeling targets specific attributes of an environment, e.g. avatar, cameras or objects, assessing how a malicious user could access and manipulate these attributes. This helps identify vulnerabilities where the need-to-know principle is violated or where tampered state updates are accepted. However, our approach does not cover all possible functionalities of the platforms we tested. For example, we do not assess vulnerabilities in functionalities such as uploading custom compressed 3D objects or chat systems, which may be susceptible to issues like XSS, SQL injections, unrestricted file uploads or denial-of-service attacks from uncontrolled file decompression. These functionalities remain untested and are outside the scope of our current methodology.

Operating within the Browser. Our tool operates within the browser by inspecting JavaScript objects in the browser's engine, which imposes limitations on the size of memory that can be analyzed. A significant limitation could surface when duplicating and comparing the global object of larger applications, which could contain millions of objects. This process can overwhelm the main JavaScript thread, potentially causing timeouts and resulting in users being disconnected for inactivity. To address this, one might consider alternative methods such as using web workers or promises to keep the main thread responsive. Alternatively, conducting the analysis outside of the browser, directly within the browser's memory address space, could circumvent some of these limitations.

8 Related Works

Recently, Garrido et al. [11] proposed a comprehensive threat model for social virtual reality applications, identifying other users as potential adversaries who receive data streams. However, this model does not address the possibility of compromising these streams or client-side data structures, thereby limiting the scope of potential

attackers to fingerprinting or inferring sensitive information. Indeed, substantial works [16, 17, 19] focused on the privacy aspects of virtual reality, primarily concerning the identification and profiling—or fingerprinting—of users and their environments. These studies highlight ongoing concerns about data privacy within VR platforms. On the security front, researchers [29] have shown that exploits based on known vulnerabilities in other systems, such as XSS, are still viable in these virtual environments. More recently, efforts have shifted toward exploring intrinsic vulnerabilities unique to shared virtual worlds. For instance, Slocum et al. [26] developed a threat model specifically for shared states in augmented reality. Their work analyzes how read and write functions can be tampered with to manipulate data, allowing unauthorized access to otherwise inaccessible holograms or to place holograms in restricted areas. Similarly, Cheng et al. [6] explore the UI libraries for developing augmented reality attacks that leverage the rendering properties of the UI to, for example, hide an object or create a clickjacking attack.

9 Conclusion

In this paper, we conducted a detailed security analysis of metaverse platforms, focusing on manipulating in-memory data structures by malicious users. We explored the security implications on three popular platforms. Our findings reveal significant vulnerabilities that could enable attacks like unauthorized audio/video surveillance or continuous user position tracking, posing substantial risks within these virtual environments. This analysis underscores the need for more robust security measures to safeguard user interactions and prevent exploitation. As metaverses gain popularity, prioritizing the security and privacy of their users is crucial to maintaining safe and trustworthy virtual communities. This work serves as a foundation for future research and development aimed at enhancing the robustness of security frameworks in the metaverse.

References

[1] Erick Bauman and Zhiqiang Lin. 2016. A Case for Protecting Computer Games With SGX. In *Proceedings of the 1st Workshop on System Software for Trusted Execution*. https://doi.org/10.1145/3007788.3007792

[2] Darrell Bethea, Robert A. Cochran, and Michael K. Reiter. 2011. Server-side verification of client behavior in online games. *ACM Trans. Inf. Syst. Secur.* (2011). https://api.semanticscholar.org/CorpusID:188945

[3] Elie Bursztein, Mike Hamburg, Jocelyn Lagarenne, and Dan Boneh. 2011. OpenConflict: Preventing Real Time Map Hacks in Online Games. In *2011 IEEE Symposium on Security and Privacy*. 506–520. https://doi.org/10.1109/SP.2011.28

[4] Peter Casey, Ibrahim Baggili, and Ananya Yarramreddy. 2021. Immersive Virtual Reality Attacks and the Human Joystick. *IEEE Transactions on Dependable and Secure Computing* (2021), 550–562. https://doi.org/10.1109/TDSC.2019.2907942

[5] IEEE VR Chairs. 2020. Online IEEE VR 2020. https://web.archive.org/web/20230620105759/https://ieeevr.org/2020/online/. (Accessed on 04/27/2024).

[6] 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. (2024). https://api.semanticscholar.org/CorpusID:263913047

[7] Minyeop Choi, Gihyuk Ko, and Sang Kil Cha. 2023. BotScreen: Trust Everybody, but Cut the Aimbots Yourself. In *32nd USENIX Security Symposium*.

[8] Robert A. Cochran and Michael K. Reiter. 2013. Toward Online Verification of Client Behavior in Distributed Applications. In *Network and Distributed System Security Symposium*. https://api.semanticscholar.org/CorpusID:5918500

[9] DEF CON. 2022. DEF CON Groups VR Events. https://web.archive.org/web/20240320214525/https://www.dcgvr.org/. (Accessed on 04/27/2024).

[10] Saba Eskandarian, Jonathan Cogan, Sawyer Birnbaum, Peh Chang Wei Brandon, Dillon Franke, Forest Fraser, Gaspar Garcia, Eric Gong, Hung T. Nguyen, Taresh K. Sethi, Vishal Subbiah, Michael Backes, Giancarlo Pellegrino, and Dan Boneh. 2019. Fidelius: Protecting User Secrets from Compromised Browsers. In *2019 IEEE Symposium on Security and Privacy (SP)*. https://doi.org/10.1109/SP.2019.00036

[11] Gonzalo Munilla Garrido, Vivek Nair, and Dawn Song. 2024. SoK: Data Privacy in Virtual Reality. *Proceedings on Privacy Enhancing Technologies* (2024). https://petsymposium.org/popets/2024/popets-2024-0003.php

[12] Kang Li, Shanshan Ding, Douglas D. McCrearyD.D. McCreary, and Steve Webb. 2004. Analysis of state exposure control to prevent cheating in online games. In *International Workshop on Network and Operating System Support for Digital Audio and Video*. https://api.semanticscholar.org/CorpusID:26210932

[13] Junsu Lim, Hyeonggeun Yun, Auejin Ham, and Sunjun Kim. 2022. Mine Yourself!: A Role-playing Privacy Tutorial in Virtual Reality Environment. *CHI Conference on Human Factors in Computing Systems Extended Abstracts* (April 2022), 1–7. https://doi.org/10.1145/3491101.3519773

[14] Meta. 2024. Explore worlds in Meta Horizon Worlds. https://www.meta.com/en-gb/help/quest/articles/horizon/explore-horizon-worlds/explore-worlds-horizon/. (Accessed on 07/18/2024).

[15] VRChat Metrics. 2024. 100k+ Concurrent Users. https://metrics.vrchat.community/?orgId=1&from=1708827218205&to=1708850191393. (Accessed on 04/29/2024).

[16] Mark Roman Miller, Fernanda Herrera, Hanseul Jun, James A. Landay, and Jeremy N. Bailenson. 2020. Personal identifiability of user tracking data during observation of 360-degree VR video. *Scientific Reports* 10, 1 (Oct. 2020), 17404. https://doi.org/10.1038/s41598-020-74486-y

[17] Robert Miller, Natasha Kholgade Banerjee, and Sean Banerjee. 2022. Combining Real-World Constraints on User Behavior with Deep Neural Networks for Virtual Reality (VR) Biometrics. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 409–418. https://doi.org/10.1109/VR51125.2022.00060

[18] Christian Mönch, Gisle Grimen, and Roger Midtstraum. 2006. Protecting online games against cheating. In *Network and System Support for Games*. https://api.semanticscholar.org/CorpusID:3067628

[19] 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. (2023). http://arxiv.org/abs/2302.08927

[20] Vivek Nair, Gonzalo Munilla Garrido, Dawn Song, and James O'Brien. 2023. Exploring the Privacy Risks of Adversarial VR Game Design. *PoPETs* (2023). https://doi.org/10.56553/popets-2023-0108

[21] Kelsey E Onderdijk, Lies Bouckaert, Edith Van Dyck, and Pieter-Jan Maes. 2023. Concert experiences in virtual reality environments. *Virtual Reality* (2023).

[22] Seonghyun Park, Adil Ahmad, and Byoungyoung Lee. 2020. BlackMirror: Preventing Wallhacks in 3D Online FPS Games. *Conference on Computer and Communications Security* (2020). https://doi.org/10.1145/3372297.3417890

[23] Ralph Schroeder. 2002. Social interaction in virtual environments: Key issues, common themes, and a framework for research. In *The social life of avatars: Presence and interaction in shared virtual environments*. Springer, 1–18.

[24] Adam Shostack. 2014. *Threat modeling: Designing for security*. John Wiley & Sons.

[25] Similaweb. 2024. Website Traffic - Check & Analyze Any Website. https://www.similarweb.com/. (Accessed on 04/29/2024).

[26] Carter Slocum, Yicheng Zhang, Erfan Shayegani, Pedram Zaree, Nael Abu-Ghazaleh, and Jiasi Chen. 2024. That Doesn't Go There: Attacks on Shared State in Multi-User Augmented Reality Applications. (2024). https://www.usenix.org/conference/usenixsecurity24/presentation/slocum

[27] Three.js. 2013. https://threejs.org/docs/#api/en/. (Accessed on 04/27/2024).

[28] 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. In *31st USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity22/presentation/trimananda

[29] M. Vondrek, Ibrahim Baggili, Peter Casey, and Mehdi Mekni. 2022. Rise of the Metaverse's Immersive Virtual Reality Malware and the Man-in-the-Room Attack & Defenses. *CoSe* (2022). https://doi.org/10.1016/j.cose.2022.102923

[30] Horizon Wolds. 2022. Meta Horizon. https://web.archive.org/web/20240426084415/https://horizon.meta.com/. (Accessed on 04/27/2024).

[31] Amir Yahyavi, Kévin Huguenin, Julien Gascon-Samson, Jörg Kienzle, and Bettina Kemme. 2013. Watchmen: Scalable Cheat-Resistant Support for Distributed Multi-player Online Games. *2013 IEEE 33rd International Conference on Distributed Computing Systems* (2013). https://api.semanticscholar.org/CorpusID:11156747

[32] Jeff Yan and Brian Randell. 2005. A systematic classification of cheating in online games. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. 1–9.

[33] Su-Yang Yu, Nils Y. Hammerla, Jeff Yan, and Péter András. 2012. A statistical aimbot detection method for online FPS games. *International Joint Conference on Neural Networks* (2012). https://api.semanticscholar.org/CorpusID:14154557