Agentic Framework for Automated Security Analysis in Multi-User Extended Reality

Mutahar Ali
University of California, Irvine
Irvine, California, USA
mutahara@uci.edu

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

Abstract

Multi-user extended reality (XR) applications are increasingly deployed in high-stakes settings such as education, healthcare, and enterprise collaboration, where groups of users interact through embodied avatars in a shared virtual space in real time. To meet strict interactive latency requirements, most consumer applications are client-authoritative, where each client computes and broadcasts its own state while the server relays it without validation, leaving security-relevant decisions to untrusted clients. A recent study showed that this design allows a single malicious participant, running a modified client on their own device, to violate five security invariants (perceptual secrecy, state integrity, safety enforcement, availability, and identity integrity), demonstrating eight attacks that compromise the security, privacy, and safety of other users across 20 consumer XR apps. As the ecosystem of multi-user XR continues to grow, systematically auditing this attack surface at scale demands an automated approach.

Therefore, in this paper, we design an agentic pipeline that, given an application package, tests each security invariant and emits the runtime instrumentation that reproduces any violation it finds. We use a general-purpose LLM coding agent with access to static analysis, dynamic instrumentation, device management, and network tools, all exposed via the Model Context Protocol. The agent first provisions the app past anti-tamper protections, localizes the enforcement logic through static and dynamic analysis with a short human-guided observation phase per app, and then synthesizes attack scripts that violate security invariants. We evaluate our methodology against a manually validated dataset of attacks against 15 Meta Quest applications. Our agent successfully instrumented 14 out of 15 apps and found violations with an average precision and recall of 0.86 and 0.90 across attacks, enabling systematic security auditing of multi-user XR applications at scale.

CCS Concepts

• **Security and privacy** → **Software and application security**; *Software reverse engineering*; • **Computing methodologies** → Machine learning approaches; • **Human-centered computing** → *Virtual reality*.



This work is licensed under a Creative Commons Attribution 4.0 International License.
XR Security '26, Austin, TX, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2951-5/26/10
https://doi.org/10.1145/3842203.3844584

Keywords

Extended Reality, Multi-User XR, Client-Side Trust, Automated Security Analysis, LLM Agents

ACM Reference Format:
Mutahar Ali and Habiba Farrukh. 2026. Agentic Framework for Automated Security Analysis in Multi-User Extended Reality. In *Second Workshop on Enhancing Security, Privacy, and Trust in Extended Reality (XR Security '26), October 26–30, 2026, Austin, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3842203.3844584

1 Introduction

Multi-user extended reality (XR) applications place participants in shared 3D environments where each user is rendered as an avatar and exchanges a continuous stream of head/hand pose, voice, and shared-object state with every other participant in real time. The category has expanded well beyond entertainment to high-stakes settings, including education and training, design and collaboration, tele-health and medical training, and remote workplaces [8, 30]. To meet the sub-20 ms perceptual budget of interactive XR [18], many multi-user XR apps rely on a client-authoritative networking model: each client computes and broadcasts its own state, while the server forwards updates without validation. Security-relevant decisions (e.g., who can hear whom, who can move which object, who is currently blocked) are therefore made on the client. A malicious participant who modifies their own client can suppress those local checks, fabricate the state they broadcast, or read perceptual streams they were never meant to see. Recent work demonstrated this concretely across 20 consumer multi-user XR apps, identifying eight attacks across five security invariants [2].

Demonstrating each of those attacks required substantial reverse engineering: recovering symbols from stripped application binaries, tracing logic that enforces each invariant through decompiled code, and writing per-application runtime instrumentation to subvert it. The workflow is effective but labor-intensive: it must be redone for each new app and may break when vendors ship updates. As the ecosystem of multi-user XR continues to grow, systematically auditing this attack surface at scale demands an automated approach.

This is challenging because no fixed program signature captures the full set of violations: the same security invariant can be enforced through different classes, methods, and runtime predicates across applications that ship heterogeneous networking SDKs and anti-tamper protections, so any automation must reason over unfamiliar code. Recent advances in tool-augmented large language models (LLMs) make this kind of semantic reasoning tractable: frontier models can now discover and exploit software vulnerabilities with increasing autonomy [3, 12].

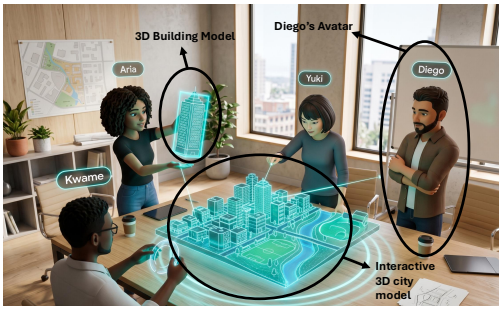


Figure 1: Overview of a multi-user XR session. Users appear as avatars with nameplates and interact with shared 3D objects in a real-time synchronized scene.

We leverage these capabilities to detect violations of high-level XR security invariants, focusing on the four invariants reproducible through runtime client manipulation. We use a general-purpose LLM coding agent, Claude Code (Opus 4.7), with access to five tool servers for static analysis, dynamic instrumentation, managed-runtime introspection, device control, and network capture, all exposed via the Model Context Protocol (MCP), and configured with a master prompt that encodes the underlying methodology and operating rules. The agent first provisions the app past anti-tamper protections, localizes the enforcement logic statically and dynamically, and then synthesizes attack scripts that violate security invariants, iteratively testing and refining them at runtime. We manually validate the attack scripts through a multi-device setup.

We evaluate the agent on 15 Meta Quest applications spanning different use cases, including social XR, gaming, education, collaboration, and medical training. The agent provisioned 14 of 15 apps for instrumentation, recovered the enforcement logic, and synthesized violating hooks, achieving an average precision of 0.86 and recall of 0.90 across six attacks. We release our framework, including prompts, scripts, and MCP tools for reuse and reproducibility [25]. In summary, we make the following contributions:

- We present, to our knowledge, the first agentic pipeline for client-side trust analysis in multi-user XR: a complete, automated workflow for testing client-side trust invariants with a tool-augmented LLM coding agent.
- We evaluate on 15 consumer Quest applications and show that the agent reproduces the prior study's per-attack findings with an average precision and recall of 0.86 and 0.90.

2 Background and Threat Model

2.1 Multi-User XR and Client-Side Trust

Multi-user extended reality (XR) applications enable groups of users to interact in a shared virtual or mixed-reality space, where each user is represented by an avatar and exchanges a continuous stream of state (e.g., head and hand poses, voice, and the state of shared objects) so that every client can render a consistent view of the scene in real time [17]. These applications are used for a variety of use cases, including education, workforce training, social interaction, and medical training simulations [8, 30]. Figure 1 provides an overview of the interaction in multi-user XR. To support safe interaction across use cases, these applications provide safety and moderation features such as muting, blocking, personal-space boundaries, and host controls that let a session owner manage who may speak, join, or interact [33]. Many apps also support user-generated content, where participants can import custom avatars and environments.

To meet the latency demands of real-time immersion [18], the majority of consumer applications adopt a client-authoritative (peer-to-server relay) communication model [2]. Unlike an authoritative server model, common in collaborative apps (e.g., Google Docs) and online games (e.g., Fortnite), where the server validates all client inputs and computes the canonical state centrally [7], multi-user XR applications require each client to compute its own state locally and broadcast it, while the server acts largely as a relay that forwards updates with little or no validation [2]. This design offloads computation from the server, and with it, trust, to individual users' devices. The integrity of the shared session, therefore, rests on the assumption that every participating client is honest.

2.2 Threat Model

The client-authoritative model holds only as long as every client is honest. We consider an adversary who controls a single, ordinary participant in a shared session and runs a modified client obtained by attaching a dynamic instrumentation framework to the running application and patching its behavior in memory. The adversary needs no privileged role, no control of the relay or the other peers, and no compromise of the server; only an ordinary account and local access to its own client.

Because the relay and the other peers act on whatever a client reports, a modified client grants the adversary three capabilities. (i) It can fabricate the state it broadcasts, so that peers act on positions, ownership, or identities that were not legitimately produced. (ii) It can suppress the local checks that enforce the room and moderation policy, exercising actions or transmitting data that the policy was meant to withhold. (iii) It can observe streams delivered to its client that it is not authorized to perceive, reading them from memory before any client-side filtering applies.

3 Motivation

To understand the security impact of the client-authoritative design, a recent work analyzed 20 consumer XR applications spanning education, workforce training, healthcare, design, and social interaction [2]. The study demonstrated eight attacks across five security invariants, i.e., conditions that must hold for a shared XR session to be safe and correct, summarized in Table 1. To do so, it developed an invariant-guided methodology that combines static binary analysis of stripped Unity IL2CPP binaries with dynamic instrumentation to locate enforcement logic and confirm each violation at runtime, demonstrating that these attacks are deterministic and reproducible across platforms and app versions.

While this methodology is effective, its reach is currently limited by the cost of applying it: identifying the enforcement logic for each invariant requires expert reverse engineering of each new application, and the resulting hooks must be rebuilt as targets evolve. Since the same categories of violation recur across apps (e.g., outbound pose serializers enable surveillance, ownership-transfer callbacks

Table 1: Client-side trust violations in multi-user XR: the five security invariants and the attacks that break them, demonstrated across 20 consumer applications [2].

Invariant	Attack(s)	Attack Description
I1 Perceptual secrecy	Video, audio surveillance	See and hear others beyond spatial limits
I2 State integrity	Object manipulation	Forge unauthorized updates to shared objects
I3 Safety enforcement	Host-control evasion, block evasion	Evade blocking and host moderation
I4 Availability	Avatar crashing, forced disconnection	Crash nearby clients or eject a user
I5 Identity integrity	Avatar cloning	Impersonate a user with a cloned avatar

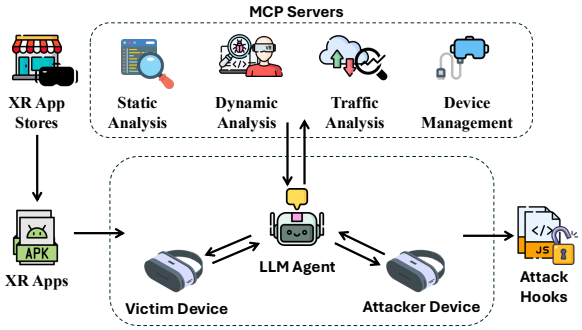


Figure 2: Overview of our methodology.

enable tampering, block-list predicates enable safety circumvention), the analysis that uncovers them should not have to be rebuilt each time. Therefore, in this work, we ask whether this end-to-end analysis for exploiting client-side vulnerabilities in multi-user XR can be automated: *given an XR application, can a system recover the necessary symbols, reason about unfamiliar code, and synthesize the instrumentation that reproduces each invariant violation, with minimal human guidance?*

Automating this analysis is non-trivial. Unlike vulnerability classes with a fixed program signature (e.g., a memory-safety bug), client-side trust violations are semantic. The same invariant can be enforced through different classes, methods, and runtime predicates across applications, anti-tamper protections vary widely, and a violation is confirmed only by acting in a live, multi-client session. The agent must therefore both reason over unfamiliar code and operate a live system end-to-end.

4 Methodology

To automate client-side security analysis in multi-user XR, we design an agentic pipeline that, given a multi-user XR application, tests each security invariant in scope (I1–I4 of Table 1)[1] and outputs the instrumentation that reproduces any violation it finds. Figure 2 shows an overview of our methodology.

Design Overview. The input is an XR application package (e.g., an Android APK or a PC VR build); the output is one of two outcomes for each attack: (i) a confirmed violation with the attack script (runtime instrumentation) that reproduces it at runtime, or (ii) no violation found. A general-purpose LLM coding agent, Claude Code (Opus 4.7), acts as the orchestrator. We give it an interface to the target through five Model Context Protocol (MCP) tool servers: a *static* server driving a headless Ghidra [21] project for symbol search and decompilation; a *dynamic* server driving Frida [11] for spawning the app, injecting code, and capturing runtime logs; an *IL2CPP* (Unity's scripting backend) server [4] that resolves and then reads, writes, or invokes classes and fields on live managed objects; a *device* server wrapping Android Debug Bridge (adb) for device management; and a *network* server that proxies HTTP(S) to expose the app's REST traffic. Together, these let the agent observe the app's logic, runtime behavior, and network traffic. The agent drives two clients running the app—an attacker and a victim—so that it can apply a change on one and observe its effect on the other. It works through a four-step pipeline per app—enabling instrumentation, static analysis, dynamic analysis, and runtime instrumentation.

Provisioning the App for Instrumentation. The agent fingerprints the platform, extracts the application binary and metadata (the inputs to static analysis), injects the Frida gadget that embeds the instrumentation runtime inside the target process, and reinstalls the app. Because many apps ship anti-tamper protections that reject a naively patched binary, the agent follows an escalating ladder of injection strategies, from plain gadget injection through anti-tamper-aware variants such as renaming the gadget binary, swapping its listen port, and applying byte-level patches. After each attempt it reinstalls, relaunches, and checks that the instrumented app starts cleanly, retrying until it launches under instrumentation or a bounded attempt budget is exhausted.

Static Analysis. Through the Ghidra MCP server, the agent issues a fixed set of queries that fingerprint the app's architecture: its networking and voice SDKs, its avatar pipeline, and the predicates and handlers that implement moderation (block, kick, mute, lock). Decompiling the hits, it narrows each invariant to a small set of candidate methods—e.g., the outbound pose publisher for perceptual secrecy, or the receive-side kick handler for safety evasion.

Dynamic Analysis. The agent loads a runtime logger that inventories the live class graph and confirms the static candidates are actually reached during a session. To resolve which methods and network events carry which behavior, it runs a short observation window in which a human operator performs a few tagged actions in-headset—moving, grabbing a shared object, blocking a test account, invoking a host control—so the agent can associate each action with the code it triggers. It also captures network traffic during this time to observe shared state and synchronization logic.

Runtime Instrumentation. For each invariant, the agent locates and instruments the method that enforces or publishes the relevant state. Every such instrumentation reduces to one of four primitives applied to a single resolved method: (i) rewrite its arguments, (ii) rewrite its return value, (iii) read or write fields on the calling object, or (iv) replace its body. The agent never writes free-form

[1]The two content-based attacks, avatar crashing and avatar cloning, require crafting and uploading malicious user-generated content and are out of scope.

exploit code; it chooses a primitive and its parameters, adapting one of four reference templates we provide. We drive the agent with a layered prompt to provide the invariants, the four primitives, the pipeline, and the operating rules. A per-phase prompt then scopes each stage of the pipeline, and a per-invariant prompt specializes the search and instrumentation for the attack under test. Having identified a candidate, the agent adapts the matching primitive template and deploys the hook through the dynamic server.

5 Evaluation

We evaluate our methodology on consumer multi-user XR apps, asking two questions: (i) how reliably can the agent provision an arbitrary commercial app for instrumentation, including bypassing the anti-tamper protections that many ship; and (ii) for the four invariants in scope (I1–I4), how accurately can it identify the enforcement logic and synthesize a hook that violates it?

5.1 Experimental Setup

Implementation Details. We use Claude Code as the LLM orchestrator with the Opus 4.7 model and a 1M-token context window. The agent is configured with the master prompt that establishes the threat model, the four manipulation primitives, the phased workflow, and the operating rules. We use publicly available MCP servers as the agent's tool surface (e.g., GhidraMCP [14] for headless Ghidra access; `frida-mcp` [6] for dynamic instrumentation). We add our own scripts and supporting documentation for the use cases each server does not cover out of the box (e.g., a patch that strips PairIP, Google Play's anti-tamper wrapper, and a delayed-injection wrapper for apps that scan their library list at startup). The attacker and victim run on two distinct Meta Quest devices, a Quest 3 and a Quest Pro, joined to the same private session of the target app. The orchestrator and all MCP servers run on a macOS host that reaches both headsets via `adb` over USB. We performed human interaction with each app for approximately 5–10 minutes during the dynamic-analysis phase, triggering specific actions including walking around the world, grabbing shared objects, blocking and unblocking a test account, and invoking any host-moderation control (mute, kick, seat lock, summon) the app exposes.

Apps. We evaluate on 15 multi-user XR apps that ship a Meta Quest build and were active during our study window; we excluded apps discontinued in that period (e.g., Rec Room). The selected apps span different use case categories, including social hangouts (e.g., VRChat), gaming (e.g., *Gorilla Tag*), education and workforce training (e.g., *EngageVR*, *VictoryXR Labs*), design and collaboration (e.g., *Spatial*, *ShapesXR*), and tele-health and medical training (e.g., *3D Organon XR*, *Alcove*). Together, the apps cover all six attacks across the four security invariants in our scope.

Attack Validation. Validation is manual: the human operator acts as both the victim and the attacker and confirms whether the intended effect manifests in the app—a frozen avatar, a forged object update, an ignored kick or host command. A cell in the agent's output is a true positive when the agent both claimed a violation and the human operator confirmed the expected side effect at runtime; otherwise, it is marked as a false positive. False negatives are cells where the underlying study confirmed a violation that the agent missed, typically because it did not converge on the correct enforcement method within its budget.

Table 2: Per-attack agent performance across the 14 instrumented apps, against the verdicts in the underlying study [2]. Overall is macro-averaged.

Inv.	Attack	Acc.	Prec.	Rec.	F1
I1	Video surveillance	1.00	1.00	1.00	1.00
I1	Audio surveillance	0.93	1.00	0.93	0.96
I2	Object manipulation	0.79	1.00	0.79	0.88
I3	Block evasion	0.79	0.57	1.00	0.73
I3	Host-control evasion	0.93	1.00	0.86	0.92
I4	Forced disconnection	0.71	0.57	0.80	0.67
	Overall	0.86	0.86	0.90	0.86

5.2 Results

Instrumentation Enabling and Static Analysis. The agent successfully provisioned 14 of the 15 apps for instrumentation, with VRChat being the only failure. The successful apps required a mix of provisioning strategies, including plain gadget injection with re-signing (the default), renaming the gadget binary to evade coarse library-name string scans, and reconfiguring the gadget port and mode for apps that hardcode checks against the default listener. For VRChat, the agent could not get past the integrity protections to run the app with the Frida gadget injected. It stopped after multiple failed attempts and emitted a diagnostic failure record rather than continuing to retry. Static analysis also succeeded on all apps except VRChat, which required a rooted device or live in-memory dumping to extract metadata information due to its encrypted `global-metadata.dat`, the file from which class, method, and field names are recovered in Unity IL2CPP binaries. Recovering its symbols requires either dumping the decrypted metadata from memory at runtime on a rooted device or instrumenting on a headset without hardware attestation (e.g., HTC Vive) [2].

Attacks and Validation. For each app, the agent tries to write hooks for the six attacks, each corresponding to one of the four invariants. Table 2 reports the per attack class accuracy, precision, recall, and F1 against the ground truth [2].

The agent performs strongest on the two surveillance attacks (I1), where the enforcement surface is highly stereotyped: outbound pose serializers and Unity `AudioSource` configuration methods are named consistently across apps, and the four manipulation primitives map directly to the corresponding hook templates.

Object manipulation (I2) is highly dependent on the dynamic analysis and on the human triggering the right actions (creating, grabbing, and moving shared objects) during that phase. It is tractable for apps using Photon [22], a commercial networking engine widely used for Unity multiplayer, because the agent inherits the same `RaiseEvent` skeleton.

Host-control evasion (I3) is reliable on apps with a centralized dispatcher (e.g., a single SessionRPCs class) that groups all moderation handlers in one place, where the agent's standard search returns the full handler set in a single pass. Some apps distribute it across multiple components, which makes it harder to analyze

and required the agent to retry with a broader search query before converging on the full handler set. Block evasion (I3) has the lowest precision (Table 2). The agent confuses three primitives that look syntactically similar in the dump: the block enforcement for rendering the attacker invisible to the victim, the reciprocal check, and the mute function. It therefore overpredicts, lowering precision.

Forced disconnection (I4) shows a bimodal outcome. On apps with an exposed kick RPC, the agent finds it quickly via the standard `Send_Kick`/`RPC_Kick` search pattern. On Photon-based apps where the attack instead requires exploiting master-client election, the agent can identify the relevant APIs and confirm election is enabled, but validation requires more semantic understanding of the app. For example, in some cases, such as EngageVR, the corresponding logic for master client selection is there but never triggered since the host always assumes the role of the master client.

6 Discussion

Key Takeaways and Recommendations. These results suggest that systematic analysis of client-side trust violations does not have to be a manual, per-app exercise. Once the methodology is encoded into a structured agent prompt and connected to standard instrumentation tools through MCP, the analysis becomes a repeatable workflow that anyone with the same toolchain can rerun against a new app, a new app version, or a new networking SDK. This enables three concrete deployments. First, app vendors can use the agent as a security-regression test before each release: re-running the per-invariant playbooks against a new build catches when a previously-defended primitive (e.g., a server-bound kick) regresses into a client-side check. Second, app stores can incorporate it into their submission review pipeline to flag apps whose client-side trust posture diverges from documented platform expectations, which is particularly relevant for high-stakes deployment contexts (education, healthcare, enterprise meetings) where these violations cause the most harm. Third, the workflow also lowers the barrier for third-party audit work: researchers without app-specific expertise can produce comparable results to a hand-built reverse-engineering effort, which is the prerequisite for industry-wide accountability on client-trust failures.

Cost and Scalability. The agent's end-to-end run on an app averages approximately 60–90 minutes, of which most is autonomous work (provisioning for instrumentation, static analysis, dynamic probing, and hook generation) and a small part is the human-in-the-loop validation in the headset and the interaction during dynamic analysis. Token consumption averages 1–2 million input tokens per app, dominated by Ghidra decompilation results returned through the static MCP server. For comparison, the corresponding manual workflow would take a significantly longer amount of time and resources. Cost also drops sharply on the second and subsequent apps that share a networking SDK family, since the static-survey queries and hook templates carry over directly; in our corpus, the Photon-based apps were consistently the cheapest after the first.

Limitations and Future Work. Our evaluation covers a single hardware platform (Meta Quest) and a single LLM (Opus 4.7). Reproducing the same per-invariant results across PC VR and Android mobile clients is left to future work. Several directions would sharpen the result. We do not compare against a non-LLM baseline; signature matching over recovered metadata with the same hook templates would isolate the agent's contribution. Ablating each stage would similarly show what static analysis, dynamic probing, and the layered prompt each contribute. Scripting the tagged in-headset actions would remove the human from the dynamic phase, and fine-tuning smaller models per stage could cut cost well below a frontier orchestrator.

Our methodology failed for VRChat on step 1. Reaching VRChat would require integrating a stealth-patched Frida build for the instrumentation step and, for the static phase, dumping the decrypted IL2CPP metadata from memory at runtime on a rooted device; both are off-the-shelf capabilities that the agent's current strategy set does not yet invoke and that we plan to add. For the per-attack cells where the agent disagreed with the underlying study, they are correctable with bounded prompt or tool refinements; none require fundamental changes to the agent's design. In every case, the agent's output is auditable: the per-app working directory contains a chronological MCP call log, the runtime captures, the hook scripts it deployed, and the per-attack reasoning. Disagreements with the underlying study are localized to a specific phase and method, which makes the failure-mode triage straightforward.

7 Related Work

Security and Privacy in XR. Security and privacy research on XR has extensively explored side channel attacks and inference from motion and sensor data [1, 10, 20, 23], bystander and physical-environment privacy [5], user authentication in immersive settings [26], and security of UI properties in XR [15, 19]. A separate line of work has focused on analyzing XR apps through static, dynamic, and network analysis to identify security gaps and vulnerabilities [13, 29, 34]. For example, Zuo et al. perform static binary analysis to identify vulnerable in-app purchasing implementations [34]. We also analyze app binaries and runtime behavior, but our task is much broader. Detecting and subverting five distinct security invariants across heterogeneous apps has no single pattern to encode that a fixed pipeline can capture [2], requiring us to use an LLM agent for reasoning over unfamiliar code.

Recent work has begun to explore security in multi-user MR settings, including sensitive inference in multi-user environments such as de-anonymization of avatars and keystroke inference from unencrypted motion traffic and avatar views [27, 31], shared-state manipulation attacks in cloud-AR localization systems [24], and privacy and integrity threats from a malicious participant in web-based metaverses using JavaScript memory diffing via browser developer tools [16]. Most relevant, a recent study analyzed 20 consumer XR applications and identified a family of eight attacks, including surveillance, tampering, disruption, circumvention of safety features, and impersonation [2]. We build on it by asking whether the analysis that uncovers these failures can itself be automated. Our approach is enabled by recent progress in tool-augmented LLM agents for program understanding and reverse engineering.

LLMs for Automated Vulnerability Analysis. A growing body of work applies large language models to offensive security. LLM agents have been shown to autonomously compromise web applications [9], exploit real-world one-day vulnerabilities from their

CVE descriptions [3], exploit zero-day vulnerabilities [3, 12], surface bugs through fuzzing [32], and to recover and reason about binaries [28]. However, this work largely focuses on well-defined vulnerability classes, such as memory-safety bugs, injection flaws, or specific CVEs. By comparison, client-side trust violations have no fixed program signature, so the analysis must reason over unfamiliar code to locate app-specific enforcement logic.

8 Conclusion

We presented an automated pipeline that recasts the expert-driven reverse-engineering workflow for analyzing client-side trust violations in multi-user XR as a repeatable, agent-driven workflow. A tool-augmented LLM coding agent, connected to five MCP servers for static analysis, dynamic instrumentation, managed-runtime introspection, device control, and network capture, reproduces the per-attack findings of a recent study. We evaluated our methodology on 15 Meta Quest apps to show that it works with good accuracy and produces fully auditable logs for deeper analysis. This shifts client-side trust analysis from an expert-bound, per-app cost into a repeatable workflow that vendors can use as a pre-release security-regression test, app stores can use as part of submission review, and researchers can use to scale audits.

Ethics

All experiments were run in private sessions on accounts we control, with no human subjects and no interaction with real users or their data. The agent reproduces attacks already documented in prior work [2]; it discovers no new vulnerabilities. We still report vulnerabilities to the affected vendor through responsible-disclosure channels. We do not release the per-application hooks or other operational artifacts the agent produces to prevent misuse.

Acknowledgments

This work has been partially supported by the National Science Foundation (NSF) under award 2544561. Any findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the NSF.

References

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

[2] Mutahar Ali and Habiba Farrukh. 2026. Relay and Betray: Exploiting Client-Side Authority in Multi-User Mixed Reality. In *USENIX Security Symposium*.

[3] Anthropic. 2026. Assessing Claude Mythos Preview's Cybersecurity Capabilities. https://www.anthropic.com/research/mythos-preview. Accessed: 2026-06-19.

[4] axhlzy. 2026. il2cpp-frida-mcp: Model Context Protocol Server for IL2CPP Runtime Introspection. https://github.com/axhlzy/il2cpp-frida-mcp. Accessed: 2026-08-31.

[5] Matthew Corbett, Brendan David-John, Jiacheng Shang, Y Charlie Hu, and Bo Ji. 2023. Bystandar: Protecting bystander visual data in augmented reality systems. In *International Conference on Mobile Systems, Applications and Services (MobiSys)*.

[6] dnakov. 2025. Frida MCP. https://github.com/dnakov/frida-mcp. Accessed: 2026-08-31.

[7] Edgegap. 2026. Authoritative Servers, Relays & Peer-To-Peer: Understanding Networking Types and Their Benefits for Each Game Type. https://edgegap.com/blog/explainer-series-authoritative-servers-relays-peer-to-peer-understanding-networking-types-and-their-benefits-for-each-game-types. Accessed: 2026-08-31.

[8] ENGAGE XR. 2026. Virtual and Augmented Reality for Team Building and Onboarding. https://engagevr.io/virtual-and-augmented-reality-for-team-building-and-onboarding/. Accessed: 2026-05-24.

[9] Richard Fang, Rohan Bindu, Akul Gupta, Qiusi Zhan, and Daniel Kang. 2024. Llm agents can autonomously hack websites. *arXiv preprint arXiv:2402.06664* (2024).

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

[11] Frida. 2026. Frida: A Dynamic Instrumentation Toolkit. https://github.com/frida/frida. Accessed: 2026-08-31.

[12] Google Project Zero. 2024. Project Naptime: Evaluating Offensive Security Capabilities of Large Language Models. https://projectzero.google/2024/06/project-naptime.html. Accessed: 2026-06-19.

[13] John Y Kim, Chaoshun Zuo, Yanjie Zhao, and Zhiqiang Lin. 2025. {AUTOVR}: Automated {UI} Exploration for Detecting Sensitive Data Flow Exposures in Virtual Reality Apps. In *USENIX Security Symposium*.

[14] LaurieWired. 2025. GhidraMCP. https://github.com/LaurieWired/GhidraMCP. Accessed: 2026-08-31.

[15] Bertram Liu, Vamsi Shankar Simhadri, and Xiaokuan Zhang. 2025. Hunting Insecure UI Properties in Extended Reality. In *International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*.

[16] 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 *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[17] Microsoft. 2022. Shared Experiences in Mixed Reality. https://learn.microsoft.com/en-us/windows/mixed-reality/design/shared-experiences-in-mixed-reality. Accessed: 2026-08-31.

[18] Milvus. 2026. What Networking Challenges Are Unique to VR Multiplayer Applications? https://milvus.io/ai-quick-reference/what-networking-challenges-are-unique-to-vr-multiplayer-applications. Accessed: 2026-08-31.

[19] Chandrika Mukherjee, Reham Mohamed, Arjun Arunasalam, Habiba Farrukh, and Z Berkay Celik. 2025. Shadowed Realities: An Investigation of {UI} Attacks in {WebXR}. In *USENIX Security Symposium*.

[20] 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. In *USENIX Security Symposium*.

[21] National Security Agency. 2026. Ghidra: Software Reverse Engineering Framework. https://github.com/NationalSecurityAgency/ghidra. Accessed: 2026-08-31.

[22] Photon Engine. 2026. Multiplayer Game Development Made Easy. https://www.photonengine.com/. Accessed: 2026-08-31.

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

[24] 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. In *USENIX Security Symposium*.

[25] SPECT Lab. 2026. Security Analysis of Client-Side Authority in Multi-User Extended Reality. https://github.com/SPECT-Lab/Multi-User-XR-Security-Client-Side-Authority.

[26] Sophie Stephenson, Bijeeta Pal, Stephen Fan, Earlence Fernandes, Yuhang Zhao, and Rahul Chatterjee. 2022. Sok: Authentication in augmented and virtual reality. In *IEEE symposium on security and privacy (SP)*.

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

[28] Hanzhuo Tan, Qi Luo, Jing Li, and Yuqun Zhang. 2024. LLM4Decompile: Decompiling Binary Code with Large Language Models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[29] 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 *USENIX security symposium*.

[30] U.S. Department of Homeland Security. 2024. Virtual Reality (VR) Training Systems for First Responders Market Survey Report. https://www.dhs.gov/science-and-technology/saver/virtual-reality-training-systems-first-responders. Accessed: 2026-08-31.

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

[32] Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, and Lingming Zhang. 2024. Fuzz4all: Universal fuzzing with large language models. In *IEEE/ACM International Conference on Software Engineering (ICSE)*.

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

[34] Chaoshun Zuo and Zhiqiang Lin. 2022. Playing without paying: Detecting vulnerable payment verification in native binaries of unity mobile games. In *USENIX Security Symposium*.