USENIX Security '26 Artifact Appendix: Relay and Betray: Exploiting Client-Side Authority in Multi-User Mixed Reality

Mutahar Ali
University of California, Irvine

Habiba Farrukh
University of California, Irvine

A Artifact Appendix

A.1 Abstract

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

A.1.1 Public Component

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

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

A.1.2 Private Component

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

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

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

A.2.2 How to access

Public Component. Zenodo concept DOI:

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

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

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

A.2.3 Hardware dependencies

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

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

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

A.2.4 Software dependencies

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

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

1. Download both files from the Zenodo DOI and run the setup script (`setup.sh` on macOS/Linux, `setup.ps1` on Windows) to install all dependencies. On Windows, `setup.ps1 -StaticAnalysis` also prepares the JDK and .NET runtime that the bundled Ghidra and Il2CppDumper need.

2. Install the target app and create two free test accounts (one per client).

3. For (E3) only: obtain the private component as described in § A.2.2.

The README walks through the pipeline as ordered stages, each one command with the output to expect: install, obtain the app binary, recover symbols, decompile, verify, log at runtime, and run a hook. It also works one sanitized example hook end to end, showing how the runtime logger supplies the values that were blanked out.

A.3.2 Basic Test

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

A.4 Evaluation workflow

A.4.1 Major Claims

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

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

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

A.4.2 Experiments

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

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

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

A.5 Notes on Reusability

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

A.6 Version

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