USENIX Security '24 Artifact Appendix: When the User Is Inside the User Interface: An Empirical Study of UI Security Properties in Augmented Reality

Kaiming Cheng, Arkaprabha Bhattacharya, Michelle Lin, Jaewook Lee, Aroosh Kumar, Jeffery F. Tian, Tadayoshi Kohno, Franziska Roesner
Paul G. Allen School of Computer Science & Engineering, University of Washington

2023-10-24

A Artifact Appendix

A.1 Abstract

We include the test case code for our AR UI properties experimentation on five AR platforms, including ARCore (Android), ARKit (Swift), Hololens (Unity), Oculus (Unity), and WebXR (browser). Due to the variety of hardware required, the limitation of existing simulators, and the number of platform dependencies required to compile the Unity code, we skip functionality and reproducibility badges and provide all necessary information to be used as references.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

We do not expect any security, privacy, or ethical concerns from executing our code.

A.2.2 How to access

All of the code can be found in https://github.com/kaiming-uw/AR_UI_Security/tree/5d7deddf46c3bd949972924a7028c2dd147b15e0

A.2.3 Hardware dependencies

ARKit: Requires an iOS device with an A9 or later processor.

ARCore: Requires an Android device with support for ARCore. The full list of ARCore-supported devices can be found in https://developers.google.com/ar/devices.

Hololens: Requires Hololens 2.

Oculus: Requires a passthrough-enabled Oculus headset, such as Oculus 2, Oculus Pro, or Oculus 3

WebXR: Requires a WebXR-compatible browser (Chrome) on an ARCore-supported device.

A.2.4 Software dependencies

ARKit: We built our test cases using ARKit and RealityKit for necessary AR functionalities.

ARCore: We built our test cases using ARCore v1.32.0 for AR functionalities and Sceneform SDK v1.20.5 for 3D content rendering.

Hololens: We built our three test cases using Mixed Reality Toolkit (MRTK) 2.0 and Unity 2021.3.16f1.

Oculus: We built our three test cases using Oculus Integration SDK v44.0 and Unity version 2021.3.25f1.

WebXR: We built our test cases using Three.js and WebXR API

A.2.5 Benchmarks

None

A.3 Set-up

After clone our Github repo, there will be five folders inside the **Property Code** folder. The property result refers to Table 1 in the paper.

ARCore: Download the Android Studio from this link https://developer.android.com/studio. To run the code for the property, import each property folder into Android Studio and build on the local Android device.

ARKit: Download the Xcode from this link https://developer.apple.com/xcode/. To run the code for the property, import each property folder into Xcode and build on the local iOS device. You will need to change the developer signing to a trusted Team in order to build the application.

WebXR: To setup an environment capable of AR development using WebXR, please use a trusted local server environment. For reference, we use Web Server for Chrome https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb to host our WebXR code. After setting up the local server, enter chrome://inspect/#devices in the browser and enter the folder where the WebXR code is hosted.

Oculus : Install Unity and all necessary dependencies for Oculus. Then follow all the necessary steps in https://developer.oculus.com/documentation/unity/unity-build/

Hololens : Install Unity and all necessary dependencies for Hololens. To deploy our code on Hololens2, here are some of the basic requirements

1. Build using ARM-32
2. Build for Release
3. Open Project Using Visual Studio 2022
4. Set up Visual Studio 2022 for HoloLens (e.g., connect your HoloLens to your computer and add it as target device)

A.3.1 Installation

The reviewer can download all necessary file in https://github.com/kaiming-uw/AR_UI_Security/tree/5d7deddf46c3bd949972924a7028c2dd147b15e0

A.3.2 Basic Test

We provide instructions to build our invisibility cases on ARKit as an example.

- Launches the Harness and places an anchor in the target location.
- Launches ComponentA and verifies [*Verify(1)*] that it correctly launches, i.e., Cube 1 appears visible and registers the onclick.
- Launches ComponentB and verifies [*Verify(2)*] that it correctly launches, i.e., Cube 2 appears visible, registers the onclick, and completely blocks the view of Cube 1 from ComponentA).
- Clicks "next" to launch the next sub-experiment; observes [*Observe(1)*] whether the cube inserted by ComponentB is visible or invisible.
- Raycasts (i.e., dispatches user input) in the direction of Cube 1 (from ComponentA) and records [*Observe(2)*] which object— ComponentA's visible cube or ComponentB's invisible cube—receives the click.

A.4 Evaluation workflow

N/A

A.4.1 Major Claims

A.4.2 Experiments

N/A

A.5 Notes on Reusability

N/A

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/usenixsec2024/.