USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

AutoVR: Automated UI Exploration for Detecting Sensitive Data Flow Exposures in Virtual Reality Apps

John Y. Kim, Chaoshun Zuo, Yanjie Zhao, and Zhiqiang Lin, *The Ohio State University*

https://www.usenix.org/conference/usenixsecurity25/presentation/kim-john

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: AUTOVR: Automated UI Exploration for Detecting Sensitive Data Flow Exposures in Virtual Reality Apps

John Y. Kim Chaoshun Zuo Yanjie Zhao Zhiqiang Lin
The Ohio State University

A Artifact Appendix

A.1 Abstract

This artifact is the source code of the tool: AutoVR. AutoVR's purpose is to instrument scenes, UI, and physics events on Unity VR apps ran on the Meta Quest 2. The source code of AUTOVR is available on GitHub: https://github.com/OSUSecLab/AutoVR, additionally, the experimental results of all 366 apps available at: https://doi.org/10.5281/zenodo.15832783. To evaluate AutoVR, we have developed a model test Unity VR app, which AUTOVR can comprehensively exercise all scenes and events in. Additionally, to show AutoVR's capabilities, we have instrumented AUTOVR upon the 366 VR apps and collected network traffic during instrumentation. The results of both experiments are compared with experiments using Android Monkey, a black-box Android event testing tool, in-place of AutoVR.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Please ensure account and device usage permission has been granted by the owner of the Meta Quest 2 device as VR apps are known to collect user data and could pose a risk to the owner's privacy. Our dataset uses test accounts and test devices, and no PII data were collected from humans.

A.2.2 How to access

AUTOVR and our datasets are available at https://doi.org/10.5281/zenodo.15832783, in addition to 10 real-world Unity VR APKs to run AUTOVR with. As AUTOVR continues to be developed, we have made AUTOVR available on GitHub at https://github.com/OSUSecLab/AutoVR.

A.2.3 Hardware dependencies

AUTOVR can be run on any standard machine running Linux or MacOS. A Meta Quest 2 device with developer mode (i.e., capable of using `adb`) is required to run AutoVR. While AUTOVR does not require a rooted Meta Quest 2 device, it is recommended to use a rooted device as a significant subset of the 366 apps were run on a rooted device.

A.2.4 Software dependencies

Software dependencies for AUTOVR are detailed in the README file on AutoVR's GitHub page.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

We have provided the complete instructions on how to install AUTOVR on the README page on GitHub. For the experiments, we include two APK files in the Zenodo repository: *model-app-non-rooted.apk*, and *antmonitor-modified.apk*. We describe the purpose for each APK in the following:

- *model-app-non-rooted.apk* is the model app embedded with `frida-gadget` used in §6.1 in the evaluation of the paper.
- *antmonitor-modified.apk* is a modified version of *AntMonitor* suited for the sensitive data exposure detection experiment, detailed in section §6.2 of the paper.

To install an APK file, run the following command: `adb install <app>.apk`

A.3.2 Basic Test

Once AUTOVR and the model app ((a) or (b) from A.3.1) are installed, follow the steps listed in the README file to build and run AutoVR. Use the following configurations from the Flag Table:

- *device*: (ID of the Meta Quest 2 using `adb devices`).
- *package*: *com.osuseclab.AutoVRTest*

If the device is rooted, ensure the `-rooted` flag is set. To test functionality, the expected output should show a log containing "Starting AutoVRApp", and initialization logs showing a list of (.dll) assembly names. AUTOVR should then be exercising events on the model app until the finished log shows. The effects of the events will be reflected visually in the Meta Quest 2 device.

A.4 Evaluation workflow

A.4.1 Major Claims

Our paper presents AutoVR, a tool/framework to automatically insturment scenes, UI, and physics events on Unity VR apps. In the paper, we claim that AUTOVR can indeed exercise events by leveraging the internal binary, and be used to collect any sensitive data exposures from the app using network collection tools like *AntMonitor*. We make the following major claims:

(C1): By leveraging the internal binary, AUTOVR is capable of exercising scenes, UI, and physics events from a model Unity VR test app. This is proven by experiment (E1) whose results are shown in Table 2 in section §6.2 of the paper.

(C2): AUTOVR can be used to exercise functions that expose sensitive data from outgoing traffic in a Unity VR app. This is proven by experiment (E2) whose results are shown in Figure 9 and Figure 10 in §6.2 of the paper.

(C3): AUTOVR significantly outperforms Android Monkey in both exercising events and exposing sensitive data from a Unity VR app. This is proven by both experiments (E1) and (E2), whose results are shown in Figure 11 in section §6.2 of the paper.

A.4.2 Experiments

(E1): *[Event exercising capability using the model test Unity VR app] [5 human-minutes + 200 compute-seconds + 5MB disk]: This experiment will test* AUTOVR *and Android Monkey on a model test app, where the number of scenes and events are known. When an event executes, the test app will log the number of executions and output each scene's events in locally stored json files.*

Preparation: Ensure that a Meta Quest 2 device is available with developer mode enabled (i.e., `adb` commands can be performed for the device). Perform the installation of AUTOVR and the *model-app(-non-rooted).apk* app on the Meta Quest 2 device described in the A.3.1. Ensure no other apps are running on the device.

Execution: Run AUTOVR with the instructions listed in the README file on AUTOVR's GitHub page, with the *package* argument set to *com.osuseclab.AutoVRTest*. Note: sometimes AUTOVR can initially stall. If the initialization of the assembly (.dll) files are not shown in the output log after 5 seconds, cancel execution and rerun AutoVR.

Results: Collect the results from pulling all JSON files (.json) from the following directory within the Meta Quest device: `/sdcard/Android/data/com.osuseclab.AutoVRTest/files/`. Each JSON file is named with the following scheme: <*event-type*>-<*scene name*>*.json*, where the <event-type> indicates what type of event (i.e., UI, trigger, collision) and <scene name> being the name of the scene. The JSON file structure is keyed by the event name and valued by the number of times executed. Each physics event key is schemed in the following format: <*collider name*>|<*physics event type*>|<*the other collider to perform the event*>. The outputted results from using AUTOVR and Android Monkey should reflect Table 2 in the paper.

(E2): *[Sensitive data exposure capability using real Unity VR apps] [2 human-hour + 3 compute-hour]: This experiment will test* AUTOVR *and Android Monkey on 5 real-world Unity VR apps to compare the number of data exposures exercised by* AUTOVR *and Android Monkey. Testing all 366 Unity VR apps will take weeks of instrumentation, as such, we have scaled down the number of apps to 5 for reproducibility.*

Preparation: Install all 5 Unity VR APKs (available at https://doi.org/10.5281/zenodo.15832783) onto the Meta Quest 2 device. Additionally, install the *antmonitor-modified.apk* file from the Zenodo repository. Pull and unzip the Monkey source code: `monkey.zip` file from the Zenodo repository. Ensure no other apps are running on the device for each time you run AUTOVR or Monkey.

Execution: We have provided the script `run_with_antmonitor.sh` for both AutoVR and Monkey. `run_with_antmonitor.sh` requires three arguments:

- *device_name*: The Meta Quest 2 device name from `adb devices`.
- *package_name*: The package name of the app.
- *ssl_offset*: The SSL offset for SSL unpinning.

The SSL offsets for all 5 apps have been precomputed and are available at https://doi.org/10.5281/zenodo.15832783. For each package name listed, use the SSL offset for the *ssl_offset* argument. The following is an example command for one of the test apps: `./run_with_antmonitor.sh FakeDeviceName com.BMINC.ArtGateVR 0x76efa8`

Results: After each run of AUTOVR and Android Monkey, there will be a subdirectory called `results` which contain the network traffic triggered by the tools. The expected results should show that the AUTOVR runs extracted more bytes from the PCAP files for each package name than the Monkey runs.

A.5 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/usenixsec2025/.