USENIX Security '25 Artifact Appendix: Harness: Transparent and Lightweight Protection of Vehicle Control on Untrusted Android Automotive Operating System

Haochen Gong, Siyu Hong, Shenyi Yang, Rui Chang*,
Wenbo Shen, Ziqi Yuan, Chenyang Yu, and Yajin Zhou
Zhejiang University

A Artifact Appendix

A.1 Abstract

Harness is a lightweight framework that can transparently protect vehicle control on untrusted in-vehicle infotainment systems such as AAOS. Harness defines a minimal protection domain containing security-critical components related to car control. Leveraging hardware virtualization features, Harness isolates the domain, protecting components and sensitive interfaces within the domain from the untrusted OS. To demonstrate the feasibility of Harness, we implemented a prototype based on the Google Cuttlefish virtual platform and evaluated it on a Raspberry Pi 5 development board. This artifact provides the source code and instructions for building and running the prototype. We also provide pre-built images to ease the evaluation. For functionality, we present the main workflow of Harness via system logs and conduct an attack simulation. For reproducibility, we provide the benchmarks stated in the paper to evaluate the overhead of Harness and the scripts for processing the results.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Harness is a protection framework that does not pose any security risks. The evaluation needs to be conducted on a development board. Since we have modified the board's host kernel, which may cause some instability, we recommend backing up the existing data on the board in advance. Our attack simulation is conducted within a virtual machine and dose not affect the host. We provide an Internet-accessible board with a prepared environment to facilitate artifact evaluation. With this, reviewers do not need to worry about risks.

A.2.2 How to access

We host this artifact on Zenodo, and the DOI is 10.5281/zenodo.14723474, please refer to the latest version. During the AE period, reviewers can access our remote device.

A.2.3 Hardware dependencies

We evaluate Harness on a Raspberry Pi 5 development board with a 4-core Cortex-A76 64-bit 2.4 GHz Broadcom BCM2712 SoC and 8 GB RAM. Our artifact includes the modified AOSP components. Hence, an x86 machine is required to build the source code since AOSP can only be built on the platform. We recommend using Ubuntu 20.04 for the OS. In addition, building AOSP can be resource- and time-consuming. Please ensure the machine has over 300 GB of disk space and maintains a good network connection.

A.2.4 Software dependencies

Our prototype is developed based on Raspberry Pi kernel (rpi-6.6.y), AOSP (android-13.0.0_r41), and Android common kernel (common-android13-5.15). The Raspberry Pi kernel runs as the host kernel of the board. The host uses Google Cuttlefish (v0.9.29) to run AAOS within a virtual machine. To build the prototype, please ensure that the device has `git` and `repo` installed.

A.2.5 Benchmarks

All benchmarks we use for evaluation are provided in the artifact. Among them are two third-party benchmarks, LMBench 3.0 and Geekbench 5.5.1, which can be installed after the AAOS boot. The other benchmarks should be placed in the AOSP source tree and built into the system image.

A.3 Set-up

The Harness prototype can be divided into two parts: host and guest. The host part includes the modified Raspberry Pi kernel and a host kernel module containing the Harness Lowvisor (Section 4). The guest part includes the modified AOSP userspace framework and Android common kernel containing the Harness guest kernel module (Section 4). There are two ways to prepare the environment for evaluation. One

*The corresponding author.

is to build the prototype from scratch, and the other is to use pre-built images. If you choose the latter, please refer to the `Image Installation` section below directly. Reviewers using our remote device can download and follow the `README` from Zenodo to complete the evaluations.

A.3.1 Installation

Build from scratch.

1. *Build the host kernel.* Enter `src/harness_host/` and run `setup.sh`. The script will download, patch, and build the Raspberry Pi kernel. Please refer to the Raspberry Pi document [1] for more details.

2. *Build the guest Android common kernel.* Enter `src/harness_guest_kernel/` and run `setup.sh`. The script will automatically download, patch, and build the Android common kernel. The output images will be placed at `guest-kimage` in the directory. Please refer to document [2] for more details.

3. *Build the guest AOSP.* Enter `src/harness_aosp/` and run `setup.sh`. The script will automatically download, patch, and build the AOSP. This operation will take a few hours and the output images will be placed at `aosp-image` in the directory. Note that benchmarks in APK format are built during this time. Please refer to document[3] for more details.

4. *Build the Lowvisor.* Enter `src/harness_host/harness_host_kernel_module/` and run `make` to build the kernel module `harness.ko`.

Image Installation.

1. *Install the host kernel*. If using the prebuilt image, upload the `images/host-kernel/kernel_2712.img` to the `/boot/firmware` of the board. Otherwise, follow the Section *Cross-compile the kernel* in the document[1] to install the kernel on the boot media. Make sure that the device uses the `kernel_2712.img` as the kernel.

2. *Install guest images*. If using the prebuilt images, simply upload the `images` folder to the board and extract the AOSP archive files inside (a `.zip` and a `.tar.gz`) in need. Otherwise, upload the AOSP and Android common kernel images to the Raspberry Pi and extract the archive files. Place all the images in the same folder (the *working directory*).

Cuttlefish Installation. Upload the pre-built packages (`images/cuttlefish-prebuilt`) to the board and run the `install.sh`. Follow the Cuttlefish document[4] for details.

[1] https://www.raspberrypi.com/documentation/computers/linux_kernel.html
[2] https://source.android.com/docs/setup/build/building-kernels
[3] https://source.android.com/docs/setup/build/building
[4] https://source.android.com/docs/devices/cuttlefish/get-started

Lowvisor Installation. If using the prebuilt images, install the `.ko` file in the working directory using `insmod`. Otherwise, upload the `harness.ko` to the board and install it using `insmod`.

A.3.2 Basic Test

Enter the working directory and run:

```
HOME=$PWD ./bin/launch_cvd
-kernel_path=./guest-kimage/Image
-initramfs_path=./guest-kimage/initramfs.img
-daemon -cpus=4 -memory_mb=6144 -vhost_net=true
```

Run `./bin/adb shell` to connect to the device terminal. The guest log will be output to the `logcat` and `kernel.log` in `cuttlefish/instances/cvd-1/logs/`. Follow the Cuttlefish WebRTC document [5] to check if the virtual device is successfully booted and enters the home screen of AAOS.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *Harness can transparently enclave userspace Andorid components (e.g., apps and services) without intrusive modification. This is proven by the experiment (E1), which demonstrates the functionality of Harness enclaves through system logs.*

(C2): *Harness can defeat potential attacks and protect the vehicle control chain of AAOS. This is proven by the attack simulation experiment (E2) described in Section 6.2 of the paper.*

(C3): *The performance and memory overhead of Harness are acceptable. This is proven by the experiment (E3) described in Section 6.3 of the paper, which covers a range of microbenchmarks (e.g., LMBench) and an application benchmark (Geekbench 5).*

A.4.2 Experiments

(E1): *[Basic Functionality] [30 human-minutes]: Harness can enclave userspace Andorid components (e.g., apps and services) without intrusive modification.*
How to: Combine system logs and process status to check the workflow and usage of Harness enclaves.
Preparation: Enable Lowvisor logging in the host kernel module source, then compile and install the module.
Execution: (1) During the AAOS boot, check the host system logs using the `dmesg` command. The Lowvisor will output the creation and execution status of enclaves. (2) After booting, connect to the virtual device using `adb`, using `ps -A` command to check process states. You will see the enclaved Zygote (`zygote64_enclaved`) running, and its child processes are also enclaved.

[5] https://source.android.com/docs/devices/cuttlefish/webrtc

(3) Using the `adb install` command to install the Geekbench (`benchmarks/Geekbench 5_5.5.1_APKPure.apk`) and run it. You will see the Geekbench process is forked from the enclaved Zygote.
Results: If the observed logs and process status meet expectations, the functionality of the Harness enclave can be validated. Since Geekbench's source code cannot be modified, its successful execution inside an enclave implies that no intrusive modifications are needed.

(E2): *[Security Features] [1 human-hours]: Harness can achieve our claimed security guarantee.*
How to: Run the attack simulation and check if the Lowvisor can detect malicious behaviors.
Preparation: (1) Memory-based attacks: Apply the patch `attack_sim/mattack_gkernel.patch` to the Android common kernel. Replace the host kernel with the `attack_sim/kernel_2712-att.img` and use the host kernel module in `attack_sim/`. Only use this image for memory-based attack simulation to avoid conflicts. Enter `attack_sim/sectests` and run `make`. Using `adb push` to upload the output binaries and `do_sectests.sh` to the virtual device.
(2) Malicious IPC: The `BinderBench` and `TestAppService` (in `aosp/device/harness/apps/`) work as a client-server pair using a protected interface. You can edit their `AndroidManifest.xml` to remove one of them from its enclave. Alternatively, you can use the prebuilt APKs (`BinderBench-nencl.apk` and `TestAppService-nencl.apk`) in the `benchmarks/`.
(3) Input injection: Uncomment the definition of the macro `HNS_ATTSIM_INPUT` in LibHarness (`aosp/framework/base/libs/harness/harness.cpp`). Enable the Lowvisor logging `HNS_EVTVF_DLOG`.
Execution: (1) Memory-based attacks: Run the script `do_sectests.sh` on the virtual device and check the log of the guest and host.
(2) Malicious IPC: Run the app `BinderBench` on the virtual device and click the `AutoBench` button. Check the `logcat`, and you will see the `Exception message` error, indicating that the transaction failed.
(3) Input injection: Open an enclaved app, click the virtual device screen 10 more times through WebRTC and check the host log.
Results: The system logs will output information about detected malicious behaviors.

(E3): *[Overhead of Harness] [1 human-hour + 1 compute-hour]: Harness incurs acceptable performance and memory overhead.*
How to: Run the benchmarks; Compare the memory usage when using Harness or not.
Preparation: (1) Compile and install the LMBench: We provide the binaries of LMBench and you can simply upload the `benchmarks/lmbench-3.0-a9-eval` folder to the virtual device using `adb push`. To build it from scratch, enter the folder and run `make CC=aarch64-linux-gnu-gcc OS=linux`.
(2) Install Geekbench: Using `adb install` to install `benchmarks/Geekbench 5_5.5.1_APKPure.apk`.
(3) Non-enclaved: To evaluate the impact on non-enclaved components, install the benchmarks in `benchmarks/non-enclaved` and repeat the tests.
Execution: (1) LMBench. Enter the `src` folder of LMBench and run `env OS="linux" ../scripts/results`. The results will be saved at `results/linux/localhost.0`.
(2) Binder latency. Launch the BinderBench app and click the `AutoBench` button. The result will be output into the `logcat`.
(3) Input responsiveness. Launch the EvalApp on the virtual device and use a clicker [6] to continuously click in the blue region labeled `TEST INPUT EVENTS`. Wait for 10 15 minutes and stop the clicker. The results will be recorded in the `logcat`.
(4) Car API latency. Launch the EvalApp on the virtual device and click the red region labeled `TEST CAR API`. Then, wait for the message "EvalCarAPI: Evaluation on Car API end" to appear in the `logcat`. The results will be recorded in the `logcat`.
(5) Geekbench. Launch the Geekbench on the virtual device and run `CPU BENCHMARK`.
(6) Startup time of apps. Run `cold_start.sh` and `hot_start.sh` (placed at `benchmarks/`) and redirect the output to a file.
(7) Memory usage. Run `eval_mem.py` (placed at `benchmarks/`) in the working directory. The script will output the average memory usage during a period.
Results: Repeat these tests in the baseline system, then refer to the README to process the obtained results using the script `eval.py`. The results should be close to those presented in the paper.

A.5 Notes on Reusability

Harness is not limited to protecting vehicle control but can also be applied to secure other Android components, similar to Android Virtualization Framework[7]. Please refer to Section 6.4 of our paper for detailed discussion.

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/usenixsec2025/`.

[6] https://github.com/InJeCTrL/ClickRun
[7] https://source.android.com/docs/core/virtualization