Breaking Infrared Recapture Detection: Optical-Synthesis Attacks and Depth-Aware In-Sensor Countermeasures

Tetsu Ishizue
The University of Electro-Communications

Sara Rampazzi
University of Florida

Takeshi Sugawara
The University of Electro-Communications

A Artifact Appendix

A.1 Abstract

This artifact is related to SynthIR, an attack that evades recapture detection methods using infrared depth sensors (e.g., Scoop). It also contains a countermeasure based on recapture detection using DP sensors. For implementing SynthIR, we provide the image and depth map datasets, the scripts, and their results. The artifact for the recapture detection includes the image and depth map datasets, the scripts that perform segmentation from depth information and detect recapture, and the results obtained by feeding the datasets into the recapture detector.

A.2 Description & Requirements

A.2.1 How to access

Our research artifacts are available at: https://zenodo.org/records/19704182.

A.2.2 Licensing Restrictions

The artifact does not include the results based on CelebDFv2 (in Section 4.2, 5.1) and the synthesized DP images from NTU RGB+ (in Section 6.5) because the original dataset prohibits the publication and distribution of the datasets.

A.2.3 Data Organization

The data and scripts related to the paper that are included in the artifact evaluation are as follows:

- requirements.txt
  - Python package dependencies for environment for our paper.
- Analyzer.cpp
  - Modified version of Analyzer.cpp for Scoop Viewer.
- Attack_Feasibility.zip
  - Images and depth data for benign, simple recapture and SynthIR to evaluate feasibility in Section 4.1, corresponding Figure 3.
- Sensor_Fusion.zip
  - Images and depth data to show sensor fusion as described in Section 4.1, corresponding to Figure 4.
- Depth_First_Results.zip
  - The attack results data containing recaptured images and depth data for depth-first approach in Section 4.3, corresponding to Figure 7.
- Video_Results.zip
  - The attack results data containing a recaptured video and depth data for video evaluation as described in Section 4.4, corresponding to Figure 8.
- Printed_Images.zip
  - Recaptured images using a monitor and photo to evaluate effectiveness of printed images in Section 5.1, corresponding to Table 1.
- Recapture_Detector.zip
  - The scripts for our recapture detection pipeline as described in Section 5.3.
- Defense_Feasibility.zip
  - Depth data for real and recapture to evaluate feasibility in Section 6.4, corresponding Figure 12.
- DP_Recapture_Results.zip

– Recaptured images and depth data for DP dataset evaluaiton as described in Section 6.5, corresponding to Figure 13.

NOTE: This artifact also includes data (Image-Video_Generation.zip) and instructions related to image and video generation using the depth-first approach. The behavior of the model is probabilistic, and we do not claim this as a target of artifact evaluation. For details, please refer to the README.

A.2.4 Hardware dependencies

While no specific hardware is required, we expect a mid-range GPU to run the scripts.

A.2.5 Software dependencies

To run the experiments evaluating Scoop, the Scoop Viewer environment is necessary for (E3, E4). In addition, to compare monitor and printed images recapturing, we evaluate the image-only detectors used in Chimera for (E5). Docker is required to setup the above environments. Furthermore, for showing depth maps (E1, E2) or running our recapture detection method (E6, E7), a virtual environment with the dependencies listed in `requirements.txt` for Python 3.12.3.

A.3 Setup

Environment for Scoop. The Scoop environment is configured with the following steps. Note that we overwrite `Analyser.cpp` in the original Scoop distribution to fix a bug. See Section A.5 for details.

- First, set up the Docker container environment with the following command.

```
docker run -it --name Scoop_env --gpus all
    --shm-size=16g -w /workspace nvidia/
    cuda:12.8.1-cudnn-devel-ubuntu24.04
```

  Then copy viewer.zip (available at Scoop repository) into that environment.

```
docker cp viewer.zip Scoop_env:/workspace
```

- Set up the environment for building Scoop Viewer in the Scoop_env container.

```
apt update && upgrade
apt install -y build-essential cmake git
    unzip pkg-config python3 python3-
    colorama libopencv-dev libopencv-
    contrib-dev libpcl-dev libboost-all-dev
     libeigen3-dev
cd /workspace/
unzip viewer.zip
cd /workspace/scoop-main
mv src/Analyzer.cpp src/Analyzer_original.
    cpp
```

- In the host environment, copy the necessary files from the host environment to the Scoop_env container.

```
docker cp Depth_First_Results.zip Scoop_env
    :/workspace/scoop-main
docker cp Video_Results.zip Scoop_env:/
    workspace/scoop-main
docker cp Analyzer.cpp Scoop_env:/workspace
    /scoop-main/src
```

- In the Scoop_env container, build Scoop.

```
cd /workspace/scoop-main}
sed -i 's/\bsudo[[:space:]]\+//g' scripts/
    install_required_libraries.sh
bash scripts/install_required_libraries.sh
    install
cmake .
make -j$(nproc)
unzip Depth_First_Results.zip
unzip Video_Results.zip
mkdir results
```

Environment for Image-Only Detectors. Set up the environment by following the Chimera artifacts. Any method is acceptable as long as it produces a working environment. Here, we describe the method we used to set up the environment.

- First, set up the Docker container environment with the following command.

```
docker run -it --name Chimera_env --gpus
    all --shm-size=16g -w /workspace nvidia
    /cuda:12.8.1-cudnn-devel-ubuntu24.04}
```

  Then, copy Chimera-main.zip (available at Chimera repository) into that environment.

```
docker cp Chimera-main.zip Chimera_env:/
    workspace
```

- Set up an environment for running the recapture detectors in the Chimera_env container.

```
apt update && upgrade
apt install -y ca-certificates curl bzip2
    unzip libgl1 libglib2.0-0
curl -L -o /tmp/Miniforge3.sh https://
    github.com/conda-forge/miniforge/re
    leases/latest/download/Miniforge3-Linux
    -x86_64.sh
bash /tmp/Miniforge3.sh -b -p /opt/conda
source /opt/conda/etc/profile.d/conda.sh
```

```
unzip Chimera-main.zip
cd /workspace/Chimera-main
conda env create -f environment.yml
conda activate chimera
```

- In the host environment, copy the necessary files from the host environment to the Chimera_env container. models.tar.gz is available at Chimera repository

```
docker cp models.tar.gz Chimera_env:/
    workspace/Chimera-main/recapture-
    detection
docker cp Printed_Images.zip Chimera_env:/
    workspace/Chimera-main/recapture-
    detection
```

- Prepare the model and evaluation images in the Chimera_env container.

```
cd /workspace/Chimera-main/recapture-
    detection
tar -zxvf models.tar.gz
unzip Printed_Images.zip
```

Environment for Our Paper. It is implemented in Python 3.12.3, and you need to set up a virtual environment with the dependencies listed in requirements.txt. Use the following commands to create and activate the virtual environment, then install the packages:

```
cd $DIR
python -m venv venv
(source venv/bin/activate} for Linux)
(venv/Scripts/activate for Windows)
pip install -r requirements.txt}
unzip Attack_Feasibility.zip
unzip Sensor_Fusion.zip
unzip Recapture_Detector.zip
unzip Defense_Feasibility.zip
unzip DP_Recapture_Results.zip
```

$DIR represents the working directory.

A.4 Evaluation Workflow

A.4.1 Major Claims

The artifact is provided for verifying the following seven claims (C1)-(C7).

- **(C1):** Attack Feasibility: Results demonstrating the feasibility of SynthIR. Depth maps are obtained for the benign case, simple recapture, and SynthIR. These results show that depth appears when capturing a real scene, that simple recapture estimate flat depth, and that SynthIR can inject fake depth in recapturing.

  These results correspond to Figure 3 in Section 4.1 and are demonstrated by experiment (E1).

- **(C2):** Sensor Fusion: Results showing that sensor fusion occurs on the iPhone 15 Pro. Depth maps are obtained when capturing Object A, when capturing Object B, and when the 3D scene is Object A while the RGB scene is Object B. When the 3D scene is Object A and the RGB scene is Object B, the depth estimated by the iPhone corresponds to Object B, even though the actual 3D scene is Object A. This is due to the effect of sensor fusion. These results correspond to Figure 4 in Section 4.1 and are demonstrated by experiment (E2).

- **(C3):** Depth-First Approach: Attack results showing that the Depth-First Approach evades recapture detection by Scoop. The attack data generated by SynthIR is input to Scoop, producing the following results: ToF depth, segmentation by ToF, RGB depth, segmentation by RGB, and Scoop output, where the detected region is shown in red. These results demonstrate that the Depth-First Approach can evade Scoop. They correspond to Figure 7 in Section 4.3 and are demonstrated by experiment (E3).

- **(C4):** Video Attack: Results of the video attack. By inputting the attack data for each frame into Scoop, 318 successful frames are obtained. These results show that depth can also be injected in videos. They correspond to Figure 8 in Section 4.4 and are demonstrated by experiment (E4).

- **(C5):** Bypassing Image-Only Recapture Detector: Comparison results obtained by inputting monitor-based and paper-based recaptures into image-only recapture detectors. For original (non-recaptured) images, the TNR is 89% with Twob_DCT, 89% with Twob_DWT, and 96% with MoireDet. For monitor images, the TPR is 97% with Twob_DCT, 96% with Twob_DWT, and 99% with MoireDet. For printed images, the TPR is 72% with Twob_DCT, 48% with Twob_DWT, and 37% with MoireDet. These results show that paper-based recapture can evade image-only recapture detectors. They correspond to Table 1 in Section 5.1 and are demonstrated by experiment (E5).

- **(C6):** Defense Feasibility: Results demonstrating the feasibility of our recapture detector. For each camera, the RGB depth, RGB segmentation, DP depth, DP segmentation, and V-measure values are obtained for both the benign and recapture cases in the feasibility experiment. These results correspond to Figure 12 in Section 6.3. For all cameras, namely the EOS R10, Google Pixel 3, and Google Pixel 4, the detection results are TNR = 100%, FPR = 0%, TPR = 100%, and FNR = 0%. The resulting V-measure values for the benign scene are 0.53, 0.59, and 0.72 for the Canon EOS R10, Pixel 3, and Pixel 4, respectively. For the recaptured scene, the V-measure is 0 for all devices. These results are demonstrated by experiment (E6).

- **(C7):** Defense Evaluation on DP Dataset: For each camera in the DP image dataset, the RGB depth, RGB segmentation, DP depth, DP segmentation, and V-measure values are obtained for both the benign and recapture cases. These results correspond to Figures 13 and 14(a–c) in Section 6.4. For all cameras, namely the EOS R10, Google Pixel 3, and Google Pixel 4, the detection results are TNR = 100%, FPR = 0%, TPR = 100%, and FNR = 0%. These results are demonstrated by experiment (E7).

A.4.2 Evaluation

The claims (C1)-(C7) can be verified with the following evaluation steps (E1)-(E7).

- **(E1):** This experiment shows the attack feasibility.

  [Preparation] Perform this in Environment for Our Paper. Move into $DIR path to execute the commands:

  ```
  cd $DIR
  ```

  [Execution] Run the following commands as described in the README to execute

  ```
  bash ./Attack_Feasibility/run.sh
  ```

  [Results] This experiment will show the depth maps in Figure 3. The depth maps from iPhone and Galaxy are saved in `Attack_Feasibility/results_iphone`, `Attack_Feasibility/results_galaxy`, respectively.

- **(E2):** This experiment shows the results of the experiment verifying that sensor fusion is performed in iPhone.

  [Preparation] Perform this in Environment for Our Paper. Move into $DIR path to execute the commands:

  ```
  cd $DIR
  ```

  [Execution] Run the following commands as described in the README to execute:

  ```
  bash ./Sensor_Fusion/run.sh
  ```

  [Results] This experiment will show the depth maps in Figure 4. The depth maps are saved in `Sensor_Fusion/results`.

- **(E3):** This experiment shows the attack results by Depth-first approach.

  [Preparation] Perform this in Environment for Scoop. Move into scoop-main path to execute the commands:

  ```
  cd /workscpace/scoop-main
  ```

  [Execution] Run the following commands as described in the README to execute:

  ```
  bash ./Depth_First_Results/run.sh
  ```

  [Results] This experiment shows images generated from depth maps can evade Scoop. Scoop output results are saved in `Depth_First_Results/results_*`

- **(E4):** This experiment shows the result of the video attack.

  [Preparation] Perform this in Environment for Scoop. Move into scoop-main path to execute the commands:

  ```
  cd /workscpace/scoop-main
  ```

  [Execution] Run the following commands as described in the README to execute:

  ```
  bash ./Video_Results/run.sh
  ```

  [Results] This experiment shows that SynthIR in video can evade Scoop. The execution result outputs the total number of frames in the video (318 frames) and the number of frames detected as recapture (0 frame). Scoop output results are saved in `Video_Results/results`

- **(E5):** This experiment shows the results of the comparison between monitor and printed images recapture.

  [Preparation] Perform this in Environment for Image-Only Recapture Detector. Move into scoop-main path to execute the commands:

  ```
  cd /workscpace/Chimera-main/recapture-
      detection
  ```

  [Execution] Run the following commands as described in the README to execute:

  ```
  bash ./Printed_Images/run.sh
  ```

  [Results] The expected results are described in C5.

- **(E6):** This experiment shows the feasibility of our recapture detection. The depth maps are estimated using conventional methods (Depth Pro for RGB depth and Revisiting Disparity from Dual-Pixel Images for DP depth) and the estimated depth maps are included in the repository as part of the dataset.

  [Preparation] Perform this in Environment for Our Paper. Move into scoop-main path to execute the commands:

  ```
  cd $DIR
  ```

  [Execution] Run the following commands as described in the README to execute:

  ```
  bash ./Defense_Feasibility/run.sh
  ```

  [Results] The expected results are described in C6. The depth maps, segmentation maps from RGB and DP are saved in `/DP_Recapture_Results/recap/$DEVICE/results`.

- **(E7):** This experiments shows the results of our recapture detection on the DP dataset. The depth maps are estimated using conventional methods (Depth Pro for RGB depth and Revisiting Disparity from Dual-Pixel Images for DP depth) and the estimated depth maps are included in the repository as part of the dataset.

  [Preparation] Perform this in Environment for Our Paper. Move into scoop-main path to execute the commands:

  ```
  cd $DIR
  ```

  [Execution] Run the following commands as described in the README to execute:

  ```
  bash ./DP_Recapture_Results/run.sh
  ```

  [Results] The expected results are described in C7. The histograms, depth maps, segmentation maps from RGB and DP are saved in `DP_Recapture_Results/recap/$DEVICE/results`.

A.5 Bugfix for Scoop

While setting up the Scoop environment, we update a particular file (`Analyser.cpp`) in the original Scoop implementation to fix two bugs. This section describes the details to justify our changes.

At line 199, a variable is defined without initialization:

uint32_t total_num_of_masked_ml_cluster_points;

This variable represents the total number of points in the learning-based model point cloud within the ToF cluster being analyzed. When analyzing another cluster, this variable should be initialized and recalculated, but it continues to accumulate because of missing initialization. As a result, the calculation of the ratio of the largest ML subregion and the ratio of the sum of the top two ML subregions are affected. Since these values become smaller than they actually are, the scene may be incorrectly detected as a recapture. We fix this as follows:

uint32_t total_num_of_masked_ml_cluster_points = 0;

At line 203, the loop uses a constant index instead of the loop variable, as shown below.

if (ground_truth_cloud_clusters[0].indices.size() < (size_t)(
    analyzer_config.heuristic_sub_cloud_
    percentage_threshold ∗ frame.depth_frame.
    point_cloud−>size())) {

In the check for whether the GT cluster being analyzed is smaller than the threshold for the entire point cloud, the 0-th cluster is used every time. As a result, every cluster is evaluated based on the result for the 0-th cluster. It is fixed as follows:

if (ground_truth_cloud_clusters[i].indices.size() < (size_t)(
    analyzer_config.heuristic_sub_cloud_
    percentage_threshold ∗ frame.depth_frame.
    point_cloud−>size())) {