USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Scoop: Mitigation of Recapture Attacks on Provenance-Based Media Authentication

Yuxin (Myles) Liu, Habiba Farrukh, and Ardalan Amiri Sani, *UC Irvine;* Sharad Agarwal, *Microsoft;* Gene Tsudik, *UC Irvine*

https://www.usenix.org/conference/usenixsecurity25/presentation/liu-yuxin

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: Scoop: Mitigation of Recapture Attacks on Provenance-Based Media Authentication

Yuxin (Myles) Liu
University of California, Irvine

Habiba Farrukh
University of California, Irvine

Ardalan Amiri Sani
University of California, Irvine

Sharad Agarwal
Microsoft

Gene Tsudik
University of California, Irvine

A Artifact Appendix

A.1 Abstract

Continuous advances in photo and video manipulation yield increasingly sophisticated deepfakes that greatly endanger societal perception of reality. Deepfake detection is an intuitive and natural research direction, which is unfortunately shaping up to be a never-ending arms race. An alternative promising direction is provenance assertion, which blends hardware-based secure camera design with the cryptographic means of authenticating the source of visual content and any post-processing (e.g., filters) applied to it.

This work starts by highlighting a very effective attack type, called a *recapture attack*, against all provenance-based techniques. In such an attack, the adversary displays fake content on some form of a screen (e.g., TV, projector, or computer screen) or surface (e.g., cardboard, canvas, or paper) and uses a provenance-asserting secure camera device to capture photos and videos of the displayed content.

We then introduce Scoop,[1] a systematic solution for mitigating recapture attacks. Scoop leverages state-of-the-art depth sensing technologies as well as learning-based depth estimation to detect misleading recaptures, i.e., a recaptured photo or video where the presence of a display medium is not visually identifiable.

We implement Scoop on both iOS and Android platforms (Apple iPhone 14 Pro and Samsung Galaxy S20 Plus), using their built-in depth sensors. To evaluate the effectiveness of Scoop, we construct a first-of-its-kind dataset consisting of 78 recapture attack scenarios. Our results show that Scoop achieves as high as $\approx 95\%$ accuracy on the iPhone and 74% accuracy on the Samsung phone.

A.2 Description & Requirements

There are four artifact components in our paper.

- **Viewer**: The Scoop Viewer is the main tool that we developed to detect the existence of misleading recaptures in photos/videos.
- **iOS App**: The iOS App is a mobile application that can be used to capture photos/videos that can be analyzed by the Scoop Viewer.
- **Android App**: The Android App is a mobile application that can be used to capture photos/videos that can be analyzed by the Scoop Viewer.
- **Dataset**: The dataset that can be used to evaluate systems such as Scoop.

A.2.1 Security, privacy, and ethical concerns

This artifact should not pose any threat to the evaluators machines' security, data privacy, or other ethical concerns. However, we recommend the use of VM when evaluating our artifact, which can help avoid any potential pollution on evaluators machines' environment.

A.2.2 How to access

All parts of our artifact can be found on our Zenodo page at https://doi.org/10.5281/zenodo.15611904. Each part contains a detailed README on its usage.

A.2.3 Hardware dependencies

Depending on different part of the artifact and different ways to evaluate it, various hardware might become necessary, such as a powerful GPU, a LiDAR-equipped iPhone, and an Android phone equipped with ToF sensor. As mentioned above, details can be found in each part's README.

A.2.4 Software dependencies

Depending on different part of the artifact and different ways to evaluate it, various software might become necessary, such as OpenCV, PCL, CUDA, and so on. As mentioned above, details can be found in each part's README.

[1] Scoop: **S**ecure **C**ontent **O**rigin from **O**ptical **P**roperties

A.2.5 Benchmarks

None. Although performance and storage overhead can be evaluated, they are not our main focus in the paper.

A.3 Set-up

There are three ways to test the Scoop system, we will only talk about one of them, which performs a quick overall test of our system, without the need of any special hardware. The other two ways to test our system can be found at subsection A.5 and A.6, where their detailed workflow can be found in the READMEs of our artifact.

We now provide a quick way to test out the Scoop system. Please download the Scoop Viewer (viewer.zip), which contains everything needed to conduct a quick test of Scoop.

A.3.1 Installation

The Scoop Viewer requires the following dependencies:

- C++ compiler (e.g., g++, clang++)
- CMake 3.5 or later
- OpenCV 4.5 or later
- PCL (Point Cloud Library) 1.12 or later
- Boost 1.75 or later
- Eigen 3.3 or later
- Python 3.9 or later

You may set up the dependencies yourself and modify the CMakeLists.txt file accordingly, or you can use our provided script to set up the environment automatically. Our script is tested on Ubuntu 24.04 LTS, but it should work on other Linux distributions (e.g., Fedora, Arch Linux) and MacOS (with Homebrew) as well. Assuming the viewer is now at a directory called `$VIEWER_DIR`, you can run the following commands to set up the environment:

```
cd $VIEWER_DIR
bash ./scripts/install_required_libraries.sh
```

A.3.2 Basic Test

After the installation is complete, you can run the following commands to build the Scoop Viewer:

```
cd $VIEWER_DIR
cmake .
make -j$(nproc)
```

After the build is complete, you can run the quick test with the following command:

```
cd $VIEWER_DIR
python3 ./scripts/eval.py sample_data/ 0000 9999
```

This command will run the Scoop Viewer on the sample data in the `sample_data` directory, which contains 8 sets of data provided (with 4 unique data points). Among the 4 data points, 1 is original and the rest 3 are recaptured photos (with 2 TVs and 1 projector).

A.4 Evaluation workflow

They are already mentioned above, and for detailed description or instructions, please refer to our READMEs.

A.4.1 Major Claims

(C1): A first-of-its-kind dataset for evaluating systems such as Scoop.
(C2): A functional iOS App to capture photos and videos that can be verified using Scoop's viewer.
(C3): A functional Android App to capture photos and videos that can be verified using Scoop's viewer.
(C4): A viewer that can analyze and detect visual content that can potentially mislead people into beleiving the existence of depth.

A.4.2 Experiments

The steps to conduct evaluation are already mentioned above, and for detailed description or instructions, please refer to our READMEs.

In terms of the expected duration of the evaluation, we expect an average time of 2 hours, with the majority time spent on setting up the environment. In terms of the outcomes of the experiments, we expect out system to successfully detect the misleading recapture scenes in the above quick test we provide. As for evaluation with self-produced data, we expect them to be detected properly as well in most cases, as long as the ground truth depth information is captured correctly. As for evaluation with our dataset, we expect the accuracy to match with our claims in the paper.

A.5 Test with Your Own Data

You can also test the Scoop Viewer with your own data. To do this, you can either use our iOS or Android app to capture photos/videos, or you can use your own camera to capture photos/videos. However, please make sure you follow the guidelines in the iOS/Android app repositories for correctly capturing the data or refer to the Scoop Viewer repository for the correct format of the data. For building iOS and Android apps, please refer to the respective repositories for instructions.

After you have extracted the data from your iOS/Android app, you need to generate perceived depth data for each

photo/video. Please refer to the Scoop Viewer repository for instructions on how to generate perceived depth data. You can pick any depth estimation model that you prefer, but we recommend using the ml-depth-pro model, which we included a script for using it in the Scoop Viewer repository. You may need to refer to the script to see how to generate the perceived depth data for your photos/videos and make sure the generated data is in the correct format so that the Scoop Viewer can analyze it. After you have generated the perceived depth data, you can run the Scoop Viewer on your data with the commands provided in the Scoop Viewer repository.

A.6 Test with the Dataset

You can also test the Scoop Viewer with our dataset. We provide full instructions on how to use the dataset in the Scoop Viewer repository. You can also refer to the Scoop dataset repository for more information about the dataset.

A.7 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/.