Evading Provenance-Based ML Detectors with Adversarial System Actions

Kunal Mukherjee, Joshua Wiedemeier, Tianhao Wang, James Wei, Feng Chen, Muhyun Kim, Murat Kantarcioglu, and Kangkook Jee

Department of Computer Science, The University of Texas at Dallas

A Artifact Appendix

A.1 Abstract

The artifact evaluation process is designed to validate the repeatability and usability of the results presented in the research paper "Evading Provenance-Based ML Detectors with Adversarial System Actions." The paper introduces PROVNINJA, a novel framework designed to discover adversarial samples, also known as gadgets, specifically tailored for path-based Intrusion Detection Systems (IDS) and Graph Neural Network-based IDS. The primary objective of PROVNINJA is to identify actions that can successfully evade state-of-the-art IDSs. The evaluation process comprises two main components: training and testing the IDS and generating adversarial examples to evade the IDSs. As a valuable resource, the authors provide a GitHub link that grants access to the source code, data, and scripts necessary for reproducing the results described in the paper.

By offering these artifacts, the researchers enable fellow researchers and practitioners to replicate and build upon their work in provenance-based ML detectors. The artifacts include comprehensive software, data, and scripts employed to generate the findings presented in the paper. The accessibility of the GitHub repository ensures transparency. It fosters collaboration among researchers, facilitating advancements in the domain of provenance-based ML detectors and contributing to the overall improvement of security systems.

A.2 Description & Requirements

PROVNINJA is a system for generating evasive variants of known attack chains by replacing rare system events with chains of common system events that achieve the same ends. To support the reproduction of our results, we have provided the code, sample data, models, and intermediate files required to produce evasive attacks from the evaluation. Although our artifacts make no particular assumptions about compute power, **25**GB of disk space and **16**GB of memory are required to store and run the models, data samples, and software dependencies.

A.2.1 Security, privacy, and ethical concerns

None. No destructive steps are taken, and no data is collected during the evaluation process. The sample data provided is from our local testbed environments, so real user data is not exposed.

A.2.2 How to access

Our code, sample data, and sample results can be accessed on GitHub at https://github.com/syssec-utd/provninja/releases/tag/USENIX_23. The sample data for the Supply Chain APT is available at https://drive.google.com/file/d/1Jz0ZuiZlUEZdAgqlnfmpN2_X0Cms6Sl8/view.

A.2.3 Hardware dependencies

Running our experiments requires **25**GB of storage and **16**GB of memory for the data, code, and models.

A.2.4 Software dependencies

Our code is written in Python and uses Miniconda for environment management. Miniconda can be installed from https://conda.io/projects/conda/en/latest/user-guide/install/index.html. Simply follow the installation directions for your machine architecture. Python 3.10 will be installed as part of the environment building process. Our experiments were run on Ubuntu 18. Our ShadeWatcher experiments use a Docker container; to install Docker, please follow the instructions at https://docs.docker.com/engine/install/.

A.2.5 Benchmarks

All the datasets and models required for use with this artifact are provided in the GitHub repository and Google Drive provided in A.2.4.

A.3 Set-up

These instructions assume that you have already installed Miniconda and Docker.

A.3.1 Installation

First, clone our code, data, and configuration files. Next, update, build, and activate the Conda environment.

```
$ git clone \
  https://github.com/syssec-utd/provninja \
  --branch USENIX_23
$ cd provninja
$ conda update conda
$ conda env update -n provng -f provng.yml
$ conda activate provng
```

Next, build the Docker container for the ShadeWatcher experiments.

```
$ cd intrusion-detection-system/shadewatcher
$ docker build . -t shadewatcher
```

A.3.2 Basic Test

Once the environment has been set up, you can verify that all the packages have been installed correctly by running `python test_installation.py`. This script will import all the relevant modules and will print "INSTALLATION VERIFIED" if the Conda environment has been installed correctly. Otherwise, it will print "FAIL", along with a brief exception message. If the Docker build process finishes without issue, the Docker environment is set up.

A.4 Evaluation workflow

We provide detailed instructions on how to reproduce supporting evidence for our major claims. In the README, we also provide the exact commands to run for each major component.

A.4.1 Major Claims

(C1): PROVNINJA reduces detection rates of state-of-the-art provenance-based IDS by up to 59%. This is proven by experiments (E1), (E2) and (E3), as described in section 6.4 and presented in tables 2 and 3.

(C2): PROVNINJA is able to use event frequency data to construct inconspicuous alternatives to rare events in an attack chain. This is proven by experiment (E4); the gadget chain generation process is discussed in section 4.6 and example gadget chains are presented in tables 1 and 9.

A.4.2 Experiments

(E1): *[Path-based IDS] [10 human-minutes + 1 compute-minute]: Run the trained path-based models on the original attacks and the corresponding Ninja variants.*
How to: In `intrusion-detection-system/path-based/`, run `python sigl.py` and `python provdetector.py`. Record the recall and F1 scores for the original Enterprise APT and the "Gadget" Enterprise APT. The expected results are provided in the README.md file in this directory.
Preparation: To set the working directory, `cd intrusion-detection-system/path-based`. No additional configuration beyond the Conda environment activation from A.3 is required.
Execution: Run `python sigl.py` and `python provdetector.py`.
Results: Record the recall and F1 scores for the original Enterprise APT and the "Gadget" Enterprise APT. The recall and F1 scores for the "Gadget" Enterprise APT should be significantly lower than those of the original Enterprise APT.

(E2): *[Graph-based IDS] [15 human-minutes + 5 compute-minute]: Validate the graph-based IDS on the original Supply Chain APT, then create ninja variants to evade detection.*
How to: In `intrusion-detection-system/graph-based/`, run the S-GAT and Prov-GAT drivers; observe that the weight averge recall and F1 scores are acceptably high (>=0.88). Then, run `python provninjaGraph.py` to generate adversarial examples and observe the decrease in recall and F1 score.
Preparation: To set the working directory, `cd intrusion-detection-system/graph-based`. Download the sample Supply Chain APT data from https://drive.google.com/file/d/1Jz0ZuiZlUEZdAgqlnfmpN2_X0Cms6Sl8/view and unzip it. To assist in this step, we have provided a shell script `download_sample_supply_chain_data.sh`, which will download and unzip the data (run `./download_sample_supply_chain_data.sh`).
Execution: Run `python gnnDriver.py gat -if 5 -hf 10 -lr 0.001 -e 20 -n 5 -bs 128 -bi -s` and `python gnnDriver.py gat -if 768 -hf 10 -lr 0.001 -e 20 -n 5 -bs 128 -bi` to validate the graph-based IDS. Then, run `python provninjaGraph.py` to create and evaluate the evasive attacks.
Results: From the classification reports, record the weighted average recall and F1 scores for the original Supply Chain APT and the evasive variants. The recall and F1 scores for the evasive variants should be significantly lower than those of the original Supply Chain APT.

(E3): *[ShadeWatcher] [**10** human-minutes + **20** compute-minutes]: Demonstrate that* PROVNINJA *can evade the SOTA provenance-based security detector, ShadeWatcher.*
How to: In the provided Docker container, run `shadewatcher_train.py` and `shadewatcher_eval.py` for the benign and anomalous samples with and without gadgets to observe the gadgets' impact on detection rates.
Preparation: Execute the commands in order:

```
$ docker run -it --mount type=bind,\
source="pwd",target=/data \
-e DATASET_DIR=/data \
--name shadewatcher shadewatcher
$ cd /ShadeWatcher
$ ./prepare_shadewatcher.sh
```

Execution: Run the script to start the evalaution, `run_shadewatcher_experiments.sh`, which will train and evaluate the ShadeWatcher model on the provided benign and anomalous data with and without gadgets. Finally, run `python3.6 stat_eval.py tests` to summarize the results.

Results: Observe the true positive decreases by **35%**, demonstrating PROVNINJA's ability to evade ShadeWatcher.

(E4): *[Gadget Finding] [10 human-minutes + 1 compute-minute]: Using some sample frequency data, create high-regularity gadgets for `winord.exe` executes `notepad.exe`.*

How to: Run `gadget-finder/gadget-finder.py` to generate gadgets and measure their regularity scores. Observe that several usable (regularity > 0.003) gadgets are created for this event.

Preparation: `cd gadget-finder`. Once you are in the right working directory, no additional configuration beyond the Conda environment activation from A.3 is required.

Execution: Run the command: `python gadget-finder.py -i input.csv -p FrequencyDB/SAMPLE_WINDOWS_FREQUENCY_DB.csv -o output/gadgets.txt`.

Results: Read the `output/gadgets.txt` file and see that five usable (regularity > 0.003) gadgets are provided.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.