USENIX Security '25 Artifact Appendix: <Ares: Comprehensive Path Hijacking Detection via Routing Tree>

Yinxiang Tao[1*], Chengwan Zhang[*], Changqing An[1],
Shuying Zhuang[2†], Jilong Wang[3,1], and Congcong Miao[4†]

[1]Institute of Network Sciences and Cyberspace, Tsinghua University, Beijing, China
[2]Zhongguancun Laboratory, Beijing, China
[3]Quan Cheng Laboratory, 250103, Jinan, Shandong, China
[4]Tencent
tyx23@mails.tsinghua.edu.cn, x10000zhang@gmail.com,
{acq,wjl}@cernet.edu.cn, zhuangsy@mail.zgclab.edu.cn, mccmiao@163.com

A Artifact Appendix

A.1 Abstract

In the Artifacts, we have included all the source code mentioned in the Open-Science section, specifically the complete source code of our proposed path hijacking detection approach Ares. All the data used in our experiments were obtained from publicly available datasets, and we have described how to access these data in Ares' README documentation. Additionally, we have shared the raw data necessary to reproduce the results presented in our paper.

Furthermore, we have included the source code of three other methods in the shared link, as we compared Ares' performance against these approaches in the paper. The README document outlines how we obtained the source code for these three methods. To ensure these methods can be fairly compared with Ares on our datasets, we made certain modifications and additions to their source code. Detailed usage instructions are provided in the README files within each method's respective folder in the shared repository.

A.2 Description & Requirements

Since our artifacts include not only the source code of Ares but also the code for three other methods (Metis, DFOH, and BEAM), we will separately specify the dependencies required for each of the four approaches.

A.2.1 Security, privacy, and ethical concerns

There are no security, privacy and ethical concerns for evaluators while executing our artifact. All raw data used in experiments are publicly available. Our use of these data does not raise any security, privacy and ethical concerns. Additionally, the open-source code we have published is either obtainable from other public sources or has obtained consent from the original code creators.

A.2.2 How to access

The source code used in our experiments – including Ares and the methods compared with Ares – along with the raw experimental data, is publicly available at https://doi.org/10.5281/zenodo.15589806.

A.2.3 Hardware dependencies

Our experiment was conducted on a server with Intel(R) Xeon(R) Gold 5218R CPU@2.10GHz which has 80 cores.

To successfully run Ares, you will need a server with sufficient memory (better over 100GB, our experimental machine was equipped with 251GB RAM). Furthermore, to properly compare Ares with the other three approaches, you will need a GPU-equipped server with cuda for running models, as all three methods involve machine learning components. Moreover, you will need over 166GB disk space to store DFOH's database.

A.2.4 Software dependencies

Our experiment was conducted on a server running Ubuntu 20.04.1 LTS and Python 3.9.13.

Ares. To successfully run Ares, you need to install the dependencies listed in the README document, which include: Python dependencies specified in requirements.txt, The bgpdump toolkit, The BGPStream framework. The README provides direct links to installation instructions for these components. Additionally, to mitigate potential dependency conflicts (e.g., from pre-existing packages in the server environ-

ment prior to experimentation), we provide a requirements-full.txt file containing all explicitly tested package versions.
Metis. To compare Metis with Ares, you need to install Python dependencies listed in requirements file in its folder.
BEAM. To compare BEAM with Ares, you need to follow the instructions listed in its original readme document at 'BEAM/readme.md' to install dependencies.
DFOH. To compare DFOH with Ares, you will need to install Docker and Python dependencies. Instructions are included in DFOH's original readme document at `DFOH/dfoh_runner/README.md`.

A.2.5 Benchmarks

Ares. The data required by Ares in the experiments with this artifact are already documented in the 'Data' folder.
Metis. The data and model required by Metis in the experiments with this artifact are already documented in the 'Metis' folder and some of its sub-folders.
BEAM. The data and model required by Metis in the experiments with this artifact are already documented in the 'BEAM' folder and some of its sub-folders.
DFOH. The database required by DFOH in the experiments with this artifact can be obtained at https://dfoh.uclouvain.be/database.

A.3 Set-up

A.3.1 Installation

In the beginning, download all files at https://doi.org/10.5281/zenodo.15589806 and decompress all compressed files. You should receive a READ document along with five folders: Ares, Metis, DFOH, BEAM, and Data.
Ares. Instructions are included in readme document at `Ares/README.md`. To install dependencies, follow the instructions bellow:
1. Install bgpdump tool following instructions at https://github.com/RIPE-NCC/bgpdump/wiki. Note to add this tool to the system PATH.
2. Install libBGPStream following instructions at https://bgpstream.caida.org/docs/install/bgpstream.
3. Install Python dependencies using **`pip install -r requirements.txt`**.

This should lead you to run Ares. If there are any missing dependencies, you could supplement it or check `requirements-full.txt`.
Metis. Instructions are included in readme document at `Metis/README.md`. Install Python dependencies using **`pip install -r requirements`**.
BEAM. Original instructions are included in readme document at `BEAM/readme.md` and our instructions are at `BEAM/README.md`. Follow the original instructions to install dependencies. Note that if you want to use the model we provide and do not want to train one yourself, you do not need to create a virtual environment for BEAM.
DFOH. Original instructions are included in readme document at `DFOH/dfoh_runner/README.md` and our instructions are at `DFOH/README.md`. To install dependencies, follow the instructions bellow:
1. Install docker using installation guide at https://docs.docker.com/engine/install/ubuntu/.
2. Install Python dependencies using **`pip install -r requirements.txt`** at `DFOH/dfoh_runner/main`.
3. Download DFOH's database at https://dfoh.uclouvain.be/database and store it in `DFOH/DFOH_db`. For DFOH, the original data acquisition method described in its documentation may no longer be functional. As an alternative, you may need to implement a web crawler to retrieve this database. While we provide our original crawler script (`download_db.py`), please note that the script requires modifications to properly download subdirectory contents from the target database.
4. Pull docker images or build it yourself. The DFOH implementation includes functionality for pulling Docker images automatically, but to circumvent potential deployment obstacles, we recommend manually building the Docker images using DFOH's provided scripts. Specifically, you should run **`bash build_and_upload.sh`** at each directory except `dfoh_runner` (Bypass the login step so the script only builds the images without attempting to upload them).

A.3.2 Basic Test

After completing the installation, perform basic testing for each method using the following procedures.
Ares. Run **`python3 main_simulation.py 4 2`** and **`python3 main_his.py 7 03 97 50`** at `Ares` folder to check whether Ares can successfully function. If output files are generated in `Ares/running_log` and `Ares/simresults`, it means Ares is functioning fine.
Metis. Run **`python3 predict.py 4 2`** at `Metis/simulation` to check whether Metis can successfully function. If output are generated in files in `Metis/simulation/results`, it means Metis is functioning fine.
BEAM. Run **`python3 run_on_dataset.py 4 2`** at 'BEAM' folder to check whether BEAM can successfully function. If output are generated in files in `BEAM/simulation` and `BEAM/simulation_results`, it means BEAM is functioning fine.
DFOH. Run **`python3 run_on_dataset.py 4 2`** at `DFOH/dfoh_runner/main` to check whether DFOH can successfully function. If output are generated in files in `DFOH/simulation` and `DFOH/simulation_results`, it means DFOH is functioning fine.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Ares is effective against reported historical hijacking events and produces a few number of alarms. This is proven by E1 described in Section 5.1 whose results are illustrated/reported in Table 4.

(C2): Ares has better recall rate on our simulated hijacking event dataset than existing state-of-the-art methods. This is proven by E2 described in Section 5.2 whose results are illustrated/reported in Table 5-6, 11-16.

(C3): Ares comparable false positive rate with existing state-of-the-art methods on our sampled normal route change dataset. This is proven by E3 described in Section 5.2 whose results are illustrated/reported in Table 7.

(C4): Ares has lower runtime overhead than existing state-of-the-art methods. This is proven by E4 described in Section 5.3 whose results are illustrated/reported in Table 8.

A.4.2 Experiments

(E1): [20 human-minutes + 3+ compute-hour + 60GB memory]: Apply Ares to 12 collected historical events and find that it can successfully detect all events with a few number of alarms.
Preparation: Correctly obtain and position the Data folder contents from the provided artifact.
Execution: Run **`python3 run_his_script.py <start_index> <end_index>`** at `Ares` folder to detect each historical event using our used settings one by one.
Results: Results are stored at `Ares/running_log`. Run **`python3 check_alerts.py`** to obtain results presented in Table 4. The outcomes are expected to closely approximate the results reported in the paper.

(E2): [1 human-hour + 20-30 compute-hour if parallel executed (hundreds of compute-hour if not)]: Apply Ares as well as Metis, BEAM and DFOH to our simulated hijacking events to compare their recall rate.
Preparation: Correctly obtain and position the Data folder contents from the provided artifact. Ensure the dependencies for each method are installed and all of them can function.
Execution: Run all commands in `run_simulationdetect.sh` at `Ares`, `run_on_dataset.sh` at `BEAM` and `DFOH/dfoh_runner/main`, `run_predict.sh` at `Metis/simulation` respectively to perform the detection on simulated events (If your CPU has sufficient multi-core capabilities, you may attempt parallel execution of these commands to significantly reduce processing time).
Results: Run **`python3 check_dataset.py`** for each methods to get recall rate results. Note that due to involving machine learning, the results of Metis and DFOH may have slight differences when executed repeatedly. The outcomes are expected to show that Ares has better recall rate than other methods.

(E3): [20 human-minute + 1-2 compute-hour]: Apply Ares as well as Metis, BEAM and DFOH to sampled normal route changes to compare their false positive rate.
Preparation: Correctly obtain and position the Data folder contents from the provided artifact. Ensure the dependencies for each method are installed and all of them can function.
Execution: Run **`python3 main_normal.py > running_log/detect_normal.txt`** at `Ares`, **`python3 run_on_normal.py`** at `BEAM`, `DFOH/dfoh_runner/main`, and `Metis/simulation` respectively to perform the detection on normal changes.
Results: Run **`python3 check_normal.py`** for Ares and manually inspect results for other three methods to compare their false positive rate. Note that BEAM had two results for two different thresholds (details see Section 5.2). The outcomes are expected to show that all methods have comparable false positive rate.

(E4): [20 human-minute + 5-6 compute-hour]: Apply Ares as well as Metis, BEAM and DFOH to a same amount of BGP updates to compare their efficiency.
Preparation: Correctly obtain and position the Data folder contents from the provided artifact. Ensure the dependencies for each method are installed and all of them can function. Run **`getUpdate.py`** at `Data/updateData` to obtain BGP updates.
Execution: Run **`python3 eff_test.py`** at `Ares`, **`python3 routing_monitor/detect_route_change_riperis.py --collector rrc03 --year 2024 --month 10`** at `BEAM`, **`python3 my_run_daily.py --date 2024-10-01 --date_end 2024-10-02 --db_dir <absolute path of DFOH_db>`** at `DFOH/dfoh_runner/main`, **`python3 eff.py`** at `Metis/simulation` respectively to perform the comparison.
Results: Run **`python3 eff_check.py`** for Ares, Metis, BEAM and manually inspect results for DFOH to compare their runtime overhead (DFOH does not detect update by update, therefore only can be counted in total). The outcomes could vary due to running devices, but Ares' runtime overhead is expected to be lower than the other methods.

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/.