USENIX'23 Artifact Appendix: <HorusEye: A Realtime IoT Malicious Traffic Detection Framework using Programmable Switches>

Yutao Dong[1,2], Qing Li [*2*], Kaidong Wu[1,2], Ruoyu Li[1,2], Dan Zhao[2], Gareth Tyson[3], Junkun Peng[1,2], Yong Jiang[1,2], Shutao Xia[1,2], and Mingwei Xu[4]

[1]Tsinghua Shenzhen International Graduate School, Shenzhen, China
[2]Peng Cheng Laboratory, Shenzhen, China
[3]Hong Kong University of Science and Technology (GZ), Guangzhou, China
[4]Tsinghua University, Beijing, China

A Artifact Appendix

A.1 Abstract

In this artifact, we provide datasets and prototype related to our paper. Specifically, We use Python to implement iForest training and rule generation. We use P4 programming language to deploy Gulliver Tunnel on a H3C S9830-32H-H data center switch with an Intel Tofino switch ASIC, and test hardware performance. We use PyTorch to implement Magnifier and use TensorRT to implement quantization operations. We deploy Magnifier on a GeForce RTX 2080 SUPER. The artifact can reproduce all experimental results reported in the main body of the paper.

Our source code is available at https://github.com/vicTorKd/HorusEye.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

N/A.

A.2.2 How to access

We host our source code on GitHub at https://github.com/vicTorKd/HorusEye. Specifically, we use this commit for the artifact evaluation: https://github.com/vicTorKd/HorusEye/releases/tag/v1.0.1.

A.2.3 Hardware dependencies

CPU, GeForce RTX 2080 SUPER GPU (for Magnifier), H3C S9830-32H-H data center switch with an Intel Tofino switch ASIC (for Gulliver Tunnel hardware performance, optional).

*Corresponding author: Qing Li (liq@pcl.ac.cn)

A.2.4 Software dependencies

The artifact is based on Python, PyTorch, TensorRT, sklearn and other Python packages. All packages can be easily installed with pip; we provide a list of required packages in iot.yaml

A.2.5 Data Set

The extracted data set (used in the article experiment) can be downloaded at link (The compressed file needs to be extracted under the DataSets folder to get HorusEye/DataSets/Pcap).

Also, we can download the original Pcap file at link and re-do the feature extraction. For burst level feature extraction, in pcap_process packet, python files should be executed in the following order (You need to manually change the datasets path in .py files and more detail can be found in README):

1. cd pcap_process
2. python3 pcap2csv_attack.py
3. python3 csv_process_attack.py
4. python3 extract_flow_size.py

For flow level feature extraction:

1. cd HorusEye
2. python3 FE.py.

A.2.6 Models

Our model is placed in AE.py under the model folder, which implements our Magnifier model. In the repository, we also include our baseline model Kitsune.

A.3 Set-up

This section should include all the installation and configuration steps required to prepare the environment to be used for the evaluation of your artifact.

A.3.1 Installation

1. conda install -c anaconda conda-env (anaconda or miniconda is needed)
2. Clone the source code from https://github.com/vicTorKd/HorusEye.
3. conda env create -f iot.yaml
4. If the TensorRT installation fails during the above installation process, you need to install it manually (TensorRT-8.2.1.8).
5. Download extracted datasets.

A.3.2 Basic Test

After downloading the data and ensuring that the data path is correct, you can run the main function in the iForest_detect.py file, which is a simple demo of the rule generation algorithm in Gulliver Tunnel (no GPU required). After ensuring that TensorRT is installed and you have a GPU, you can run the control_plane.py file to get all the experimental results except hardware performance. If there is a programmable switch, you can compile iot_dect_waterflow8.p4 to the switch, and check the hardware performance through p4i.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *HorusEye notably outperforms the Kitsune model in most attacks on both our and public datasets, especially in low false positive scenarios. Additionally, HorusEye and Magnifier achieve comparable detection performance, while HorusEye is even slightly better at detecting most anomalies than Magnifier. This is proven by the experiment (E1) and (E2) described in section 6.7 whose results are illustrated in Table 5 and Table 10.*

(C2): *Rule generation algorithm can convert hundreds of iTrees into whitelists. Then, the whitelists can offload 76% of normal traffic on our dataset, i.e., the throughput gains are 4.13x. This is proven by the experiments (E1) in sections 6.4 and 6.9 whose results are illustrated in Table 1 and Figure 8(a).*

(C3): *Magnifier (fp32) has significantly better packet throughput than Kitsune. Additionally, quantizing Magnifier from 32-bit float (fp32) to 8-bit int (int8) brings about a 2.5x throughput gain while the TPR only slightly drops from 0.675 to 0.636 on our dataset. This is proven by the experiments (E1) and (E3) in sections 6.9 whose results are illustrated in Figure 8.*

(C4): *Gulliver Tunnel is fairly robust to three common black-box attacks: injection attack, low-rate attack and poison attack. This is proven by the experiments (E4) in section 6.8 whose results are reported in Figure 7.*

(C5): *Gulliver Tunnel deployed on the switch can reach 100Gbps and occupies very little TCAM and SRAM. This is proven by the experiments (E5) in section 6.6 whose results are reported in Table 3 and Table 4.*

A.4.2 Experiments

(E1): *[Experiment on our dataset] [30 human-minutes + 30 compute minutes + 20GB disk]: evaluate the detection performance and throughput of the model on our dataset.*
Preparation: *After cloning the source code, configure the conda environment according to "iot.yaml", download the extracted dataset zip and extract it to the root directory of the source code project, and use the model files packaged in the source code project to evaluate it directly without training.*
Execution: *To evaluate HorusEye (Magnifier (fp32) + Gulliver Tunnel), run "python control_plane.py --train False --experiment A --horuseye True" in the root of the source code project. To evaluate Magnifier (fp32) only, run "python control_plane.py --train False --experiment A --horuseye False". You can run "python control_plane.py --help" for more detailed instructions.*
Results: *For the evaluation of HorusEye, the detection performance of HorusEye, the throughput of Magnifier (fp32) and the throughput gain of Gulliver Tunnel will be displayed in the terminal and the detection performance results will also be saved in the csv file located at ". /result/HorusEye/record_attack.csv". For the evaluation of Magnifier (fp32), the detection performance and the throughput of Magnifier (fp32) will be displayed in the terminal and the detection performance results will also be saved in the csv file located at ". /result/Magnifier/record_attack.csv".*

(E2): *[Experiment on public dataset] [30 human-minutes + 30 compute-minutes + 20GB disk]: evaluate the detection performance of the model on public dataset.*
Preparation: *Same as that of (E1).*
Execution: *To evaluate HorusEye (Magnifier (fp32) + Gulliver Tunnel), run "python control_plane.py train False experiment B horuseye True" in the root of the source code project. To evaluate Magnifier (fp32) only, run "python control_plane.py train False experiment B horuseye False". You can run "python control_plane.py help" for more detailed instructions.*
Results: *For the evaluation of HorusEye, the detection performance of HorusEye will be displayed in the terminal and further saved in the csv file located at ".*

/result/Open-Source/HorusEye/record_attack.csv". For the evaluation of Magnifier (fp32), the detection performance of Magnifier (fp32) will be displayed in the terminal and further saved in the csv file located at ". /result/Open-Source/Magnifier/record_attack.csv".

(E3): *[Experiment with int8 model] [30 human-minutes + 20 computeminutes + 20GB disk]: evaluate the detection performance and throughput of the int8 model after quantizing on our dataset.*
Preparation: *In addition to the same as that of (E1), an additional configuration of TensorRT-8.2.1.8 is required.*
Execution: *To evaluate HorusEye (Magnifier (int8) + Gulliver Tunnel), run "python control_plane.py train False experiment C horuseye True" in the root of the source code project. To evaluate Magnifier (int8) only, run "python control_plane.py train False experiment horuseye False". You can run "python control_plane.py help" for more detailed instructions.*
Results: *For the evaluation of HorusEye, the detection performance of HorusEye and the throughput of Magnifier (int8) will be displayed in the terminal and the detection performance results will also be saved in the csv file located at ". /result/HorusEye/record_attack.csv". For the evaluation of Magnifier (int8), the detection performance and the throughput of Magnifier(int8) will be displayed in the terminal and the detection performance results will also be saved in the csv file located at ". /result/Magnifier/record_attack.csv".*

(E4): *[Experiment on robustness] [30 human-minutes + 10 compute-minutes + 20GB disk]: evaluate the detection performance (robustness) of Gulliver Tunnel under three common black-box attacks.*
Preparation: *Same as that of (E1).*
Execution: *Run "python control_plane.py train False experiment D horuseye False" in the root of the source code project. Note that after running this experiment, the parameters of Gulliver Tunel will be modified, and retraining is required when re-running other experiments. You can retrain by setting "train" to True, as in "python control_plane.py train True experiment A horuseye True". You can run "python control_plane.py help" for more detailed instructions.*
Results: *The detection performance of Gulliver Tunnel will be displayed in the terminal and the detection performance results of each type of black-box attacks will be saved in the csv file located at "./result/df_robust_result_robust_type.csv".*

(E5): *[Hardware performance] [30 human-minutes + 10 compute-minutes + 20GB disk]: evaluate the hardware performance of Gulliver Tunnel after deployment on the programmable switch. (optional)*
Preparation: *An Intel Tofino switch ASIC and a traffic generator (e.g., SPIRENT N11U).*
Execution: *(1) Use winscp to log in to the switch, and put p4 file into the switch, e.g., /mnt/onl/data/bf-sde-9.1.0/pkgsrc/p4-examples/p4_16_programs/iot/iot_dect_waterflow8.p4. (2) Use ssh to log in switch, cd $SDE/pkgsrc/p4-build. (3) ./configure –prefix=$SDE_INSTALL –with-tofino –with-bf-runtime P4_NAME=iot_dect P4_PATH=$SDE/pkgsrc/p4-examples/p4_16_programs/iot/iot_dect_waterflow8.p4 P4_VERSION=p4-16 P4C=p4c. (4) make. (5) make install. (6) Use a traffic generator to test forwarding.*
Results: *The resource occupation performance of Gulliver Tunnel will be displayed in $SDE/pkgsrc/p4-build/tofino/iot_dect/pipe/log. The single port forwarding performance can be shown in Traffic Analyzer.*

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926.