USENIX'23 Artifact Appendix: BotScreen: Trust Everybody, but Cut the Aimbots Yourself

Minyeop Choi[1], Gihyuk Ko[2,3], and Sang Kil Cha[1,2]

[1]*KAIST* [2]*Cyber Security Research Center at KAIST* [3]*Carnegie Mellon University*

{okas832,gihyuk.ko,sangkilc}@kaist.ac.kr

A Artifact Appendix

A.1 Abstract

BotScreen is a client-side distributed system for detecting aimbots in FPS games. In its operation, BotScreen is deployed in each client's machine and pre-processes incoming stream of FPS game data in a trusted manner (i.e., in SGX). Then, BotScreen uses a pre-trained deep learning model (SGRU) to detect aimbots in the game. This artifact includes the source code of BotScreen, the (anonymized) dataset of gameplay logs we collected for training and validation of BotScreen, and scripts for reproducing results in the paper. In the following sections, we provide step-wise instructions in order to reproduce the results in our paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Not applicable.

A.2.2 How to access

The source code of BotScreen is accessible through GitHub at `https://github.com/SoftSec-KAIST/BotScreen/tree/8ad88322f6abbcff6de1974103b275940a839028`. We also provide pre-processed dataset as well as pre-trained weights of the SGRU models used in our experiments at `https://doi.org/10.5281/zenodo.8058051`.

A.2.3 Hardware dependencies

To reproduce the results in our paper locally, a machine with at least one PyTorch-supported GPU is required. Specifically, we recommend using a GPU that has more than 11GB of VRAM. In our experiments, we used a machine equipped with 4 Intel Xeon Silver 4214 CPUs and 4 NVIDIA RTX 2080 Ti (11GB VRAM) GPU cards for training SGRU models, and used a machine equipped with an AMD Ryzen 9 5950X CPU and one NVIDIA RTX 3090 Ti (24GB VRAM) for testing the trained models.

A.2.4 Software dependencies

BotScreen is designed to run on a Linux machine, and we tested it on Ubuntu 20.04 and Ubuntu 22.04. Also, BotScreen is written in Python 3 (3.10), and it depends on packages such as `torch`, `numpy`, `pandas`, and more. Please refer to the provided `requirements.txt` for a full list of required Python packages.

A.2.5 Benchmarks

We make our anonymized pre-processed dataset, pre-trained SGRU models and pre-evaluated data available through Zenodo: `https://doi.org/10.5281/zenodo.8058051`.

A.3 Set-up

A.3.1 Installation

Our implementation of BotScreen depends on several Python packages. The required Python packages can be installed through `pip`, via running the following command:

```
$ pip3 install -r requirements.txt
```

Please also note that our scripts run via GNU Makefile. In the provided `makefile`, the default configurations for training and evaluation parameters are set.

A.3.2 Basic Test

First, download the dataset from `https://doi.org/10.5281/zenodo.8058051` and place the `data_processed` folder into the root of the source code.

Next, one can train SGRU models using downloaded dataset by running the following:

```
$ make train
```

The above command will train a total of 7 SGRU models, where each model is trained according to the 7-fold cross-validation split of the dataset. Specifically, it will produce the following files as output in the `trained_models` directory: `config.json`, and `gru_k0.pt`-`gru_k6.pt`. `config.json` saves the model parameters in JSON format, and `gru_k0-6.pt` contains the

trained weights of each SGRU model from 7-fold cross validation.

Once the SGRU models are trained, one can evaluate their effectiveness by running the following:

```
$ make eval
```

The above command will produce the following files as output in the `trained_models` directory: `eval_k0-eval_k6`. These files are pickle files containing (true label, predicted label) tuples, generated to speed up further evaluation.

A.4 Evaluation workflow

A.4.1 Set up trained SRGU models

There are two ways to obtain the trained SRGU models. One is to train new models from our dataset, and the other is to use the pre-trained model that we provide through Zenodo.

`[10 human-mins + 70 compute-hrs + 15GB disk]`
(Option 1) Train from our dataset as follows,

```
$ unzip BotScreen_data.zip
$ mv BotScreen_data/data_processed ./
$ make train
$ make eval
```

`[10 human-mins + 15GB disk]`
(Option 2) Use the provided pre-trained model. This can be done by copying `data_processed` and `trained_models` from provided artifact into the root of the source code.

```
$ unzip BotScreen_data.zip
$ mv BotScreen_data/trained_models ./
```

A.4.2 Major Claims

(C1): BotScreen can detect aimbot properly. This is proven by Experiment (E1) described in Section 5.2.1 of our paper.

(C2): BotScreen can perform better than previously suggested methods. This is proven by Experiment (E2) described in Section 5.4 of our paper.

(C3): Differences in observations does not impact largely to BotScreen. This is proven by Experiment (E3) described in Section 5.5 of our paper.

A.4.3 Experiments

(E1): `[5 human-minutes]`
The experiment will show the accuracy of the detection model.
How to: Run `make experiments/exp_bench` and see result in `bench.tsv`.
Preparation: Trained model and evaluation data in A.4.1 are needed.
Execution: `$ make experiments/exp_bench`
Results: Check the report file in `bench/bench.tsv` Each line in `bench.tsv` shows the result of model's performance from each split.

(E2): `[15 human-minutes + 30 compute-minutes]`
The experiment will compare the performance between previous tools and BotScreen.
How to: Run experiments in `comp_study` and `experiments/exp_bench`.
Preparation: Dataset, trained model and evaluation data in A.4.1 are needed.
Execution: Execute whole experiments is as follows,

```
$ make experiments/exp_bench
$ make comp_study/th_vara
$ make comp_study/th_acca
$ make comp_study/th_kill
$ make comp_study/ks_acca
$ make comp_study/os_cac
$ make comp_study/os_lac
$ make comp_study/os_smac
$ make comp_study/history
```

Results: Each experiment will print the evaluation result.

```
$ make experiments/exp_bench
```

Botscreen:
best_acc: 0.9764, best_prec: 0.9685, auc_roc: 0.9712
TP: 63, TN: 185, FP: 1, FN: 5

Experiment `comp_study/history` will produces history based detection result of each player in tsv file per methods.

```
$ make comp_study/history
$ ls *.tsv
```

history_botscreen.tsv history_os_LAC.tsv
history_th_Kill.tsv history_ks_AccA.tsv
history_os_SMAC.tsv history_th_VacA.tsv
history_os_CAC.tsv history_th_AccA.tsv

(E3): `[5 human-minutes + 20 compute-minutes]`
The experiment will show the differences in detection results between players.
How to: Run `make experiments/stat_obs`, `make experiments/exp_obs` and see the statistic of differences of observation bewteen clients and see effects of observation rate to accuracy.
Preparation: Dataset, trained model and evaluation data in A.4.1 are needed.
Execution: Run as follows,

```
$ make experiments/stat_obs
$ make experiments/exp_obs
```

Results: After running `experiments/stat_obs`, saved statistic data and visualized results of each game are stored in the `data_loss` directory.

```
$ make experiments/stat_obs
$ ls data_loss/exp_1/
```
figures game_1 game_2 ...

Next, by running `experiments/exp_obs`, you will get a figure in `figures/fig_07_obs.pdf` which is Figure 8 in the paper.

A.5 Notes on Reusability

A.5.1 How to train and evaluate under different parameters

If you want to train and evaluate BotScreen with different model parameters, you can try it by changing the corresponding parameter values defined in `makefile` such as number of hidden units, number of layers, and more. We refer to our paper for a detailed explanation on what each parameter means.

A.6 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/.