USENIX'23 Artifact Appendix: DeResistor: Toward Detection-Resistant Probing for Evasion of Internet Censorship

Abderrahmen Amich, Birhanu Eshete, Vinod Yegneswaran and Nguyen Phong Hoang

.1 Abstract

DeResistor is a research project that provides a system extension to protect Probing for Evasion of Internet Censorship from detection. Specifically, it offers IP address protection for internet users that are running automated tools for censorship measurments and evasion (e.g. Geneva).

In this artifact, we provide an instance of DeResistor implemented on top of Geneva code: https://github.com/Kkevsterrr/geneva. DeResistor leverages Machine Learning techniques to model a censor-side flow-level detector and use it to guide Geneva genetic evolution towards more detection-resilient evasion strategies. Additionally, DeResistor introduces guided-pauses of censorship evasion attempts and interleaving them with normal user-driven network activity to confuse IP-level detection.

.2 Description & Requirements

.2.1 Security, privacy, and ethical concerns

For Docker experiments, evaluators have no risk to execute this artifact. However, real-world experiments are intended to test DeResistor in censored regimes (e.g., China, India, etc). Evaluators should not try to reproduce these experiments in one of these countries using there personal machines. Instead, they need to get access to vantage points that are remotely controllable and do not require the involvement of real user credentials that may identify individuals.

.2.2 How to access

Our artifact can be accessed via https://github.com/um-dsp/DeResistor.

.2.3 Hardware dependencies

No specific Hardware dependencies are needed for docker experiments. However, real-world experiments have to be performed in censored regimes. If needed, we can provide you with `ssh` access to one of our vantage points in China.

.2.4 Software dependencies

DeResistor has been developed and tested on Ubuntu. However, it should support Centos or Debian-based systems. Similar to Geneva, due to limitations of netfilter and raw sockets, this code does not work on OS X or Windows at this time and requires python3.6. To reproduce In-situ experiments, docker has to be properly installed.

.2.5 Benchmarks

None

.3 Set-up

.3.1 Installation

- Install netfilterqueue dependencies:

```
$ sudo apt-get install build-essential
python-dev libnetfilter-queue-dev
libffi-dev libssl-dev iptables python3-pip
```

- Create a new python3.6 environment and install Python dependencies:

```
$ python3 -m pip install -r requirements.txt
```

- **If needed**, for Debian 10 systems, you can install netfilterqueue directly from Github:

```
$ sudo python3 -m pip install --upgrade -U
git+https://github.com/kti/python-
netfilterqueue
```

- **If needed**, on Arch systems, you can make liblibc.a available for netfilterqueue:

```
$ sudo ln -s -f /usr/lib64/libc.a
/usr/lib64/liblibc.a
```

- After you make sure you install and run docker on your system use the dockerfile provided in */docker* folder to build the base image:

```
$ sudo docker build -t
base:latest -f docker/Dockerfile .
```

.3.2 Basic Test

- to manually run/inspect the docker image to explore the image, run:

  ```
  $ sudo docker run -it base
  ```

- To check that all docker containers and harpoon are running correctly, We can run the genetic algorithm with small number of *Individuals* (`--population`) and *Generations* (`--generation`):

  ```
  $ sudo /path/to/python_environment/bin/python
  evolve.py --censor censor3 --server
  forbidden.org -- log debug --workers 1
  --runs 1 --population 5 --
  generation 1 --jump 1
  ```

  [To add: Output description]

.4 Evaluation workflow

.4.1 Major Claims

(C1): DeResistor offers detection-resilience to Geneva's probing traffic. This is discussed in §6.2. We record the flow-level detection rate and IP-level detection result of DeResistor and Geneva in Table 1.

(C2): Effectiveness of DeResistor to produce working strategies that can evade the censor.

.4.2 Experiments

Experiments (E1) and (E2) reproduce results related respectively to major claims (C1) and (C2). If the evaluators do not have access to controlled vantage points in China, India or Kazakhstan, in order to reproduce results in the first 3 rows of Table 1 and working strategies in Table2, they can rely on the docker experiments to reproduce results in rows 4-8 in Table 1 (addresses (C1)) and monitor the traffic logs of newly generated strategies that they have found against the mock censors to address (C2). If evaluators are able to generate strategies against real-world censors (e.g., startegies in Table2 in the paper), they can test those strategies directly against the censor using Geneva engine.

(E1): [Testing Detection-Resilience] [30 human-minutes + 1 to hours compute-hour according to the considered number of individuals and generations. 2GB disk should be sufficient to store all the results]:
We run DeResistor vs. one of the 11 mock censors, while enabling real-time detection using `--real-time-detection`. During execution time, we keep track of the *Detection Rate* after every strategy evaluation, displayed to the screen. Additionally, we check whether the real-time detector blocks the IP address and stops Geneva/DeResistor training.

How to: We start by running Geneva without DeResistor protection, using `--Geneva` to test its flow-level and IP-level detection. Then, we run DeResistor and compare the results between both runs. We provide more detailed description later in the *Execution* paragraph.

Preparation: To prepare this experiment, we need to make sure all Setup points in §3 are taken care of.
You can ignore the following if you are running only docker experiment: – For real-world experiments, You need to have internet access. For experiments in china, we need to first bypass DNS poisoning as exaplained in §6.1 in the paper, paragraph 3. For Linux systems we can point the URL `hrw.org` to its correct IP `23.185.0.2`, by adding the line `23.185.0.2 www.hrw.org` to `/etc/hosts`. Similarly, to run this experiment in Kazakhstan we need to add `93.184.216.34 www.youporn.com` to `/etc/hosts` file.

Execution:
- Running Geneva without DeResistor protection.

  ```
  $ sudo /path/to/python_environment
  /bin/python evolve.py --censor censor1
  --server forbidden.org -- log debug
  --workers 1 --runs 1 --population 200
  --generation 10 --Geneva
  --real-time-detection
  ```

  Before performing a second run make sure all docker containers related to Geneva are killed using: `"sudo docker kill $(sudo docker ps -q)"`. The execution automatically stops after testing only two flows which is a evidence of IP-level detection (reported in Table1 in the paper). To reproduce Geneva's flow-level detection value, we need to complete all Geneva training by disabling `--real-time-detection` (remove it from the command) to avoid early blocking of Geneva. After every iteration, we observe how *The detection Rate* value changes to reach ≈ 99% as reported in Table1 in the paper rows 4-8.

  For real-world experiments (e.g., China):

  ```
  $ sudo /path/to/python_environment/bin
  /python evolve.py --external-server
  --server www.hrw.org --test-type http
  --log debug --workers 1 --runs 1
  --population 500 --generation 20
  --real-time-detection
  --local-model rfc_gfw.joblib
  --censor-model rfc_gfw2.joblib
  --Geneva
  ```

  `www.hrw.org` is not necessarily censored outside of China. For India, we can use `bannedthought.net, xnxx.com, vidwatch.me` and for Kazakhstan, we can use `youporn.com`. For india and Kazakhstan experiments we can use more appropriate models provided in `/ML detectors` folder, respectively called `rfc_india.joblib` and `rfc_kz.joblib` to update `-local-model` and `-censor-model`. Before performing a second run make sure you reset the iptables: "`sudo iptables -F`".

- Running DeResistor:

  ```
  $ sudo /path/to/python_environment
  /bin/python evolve.py --censor censor1
  --server forbidden.org -- log debug
  --workers 1 --runs 1 --population 200
  --generation 10 --jump 1
  --real-time-detection
  ```

  Before performing a second run make sure all docker containers related to Geneva are killed using: "`sudo docker kill $(sudo docker ps -q)`". According to Table 1 in the paper, DeResistor should be able to complete its training without IP-level detection. Similar to the previous run, we also track the changes occurred on the *Detection Rate* as it regularly displays in the console.
  For real-world experiments, we can use the same command as before without `--Geneva` and adding *--jump 1*. Similarly, we need to flush the iptables after each run with: "`sudo iptables -F`".

To reproduce results against all 11 mock censors, we can run the same commands using a different censor (e.g., `--censor censor2`, etc).

Results: Results of every run are stored in the folder `/trials/[date-and-time-of-execution]`. It contains , network traces in `/packets` and their csv counterparts after features extraction in `/csv` (using only the client-side packets). All generated strategies are located in the final `hall.txt` file (e.g., `/generations/hall9.txt`) with their fitness values. Strategies with the highest fitness values are most likely to evade the censor.

As illustrated before, detection resilience results should be observable during run-time. Particularly, if the program raises a `detection` exception, then an IP detection is observed. Additionally, the flow-level detection rate is regularly displayed during run-time as `Detection Rate`. We also store the flow-level detection results of all flows in `preds.csv`. We can re-compute the final value of the flow-level detection rate by counting the percentage of zeroes (0: *detected as Geneva flow*) compared to all flow-level predictions.

(E2): [Evasion Effectiveness:] [1 to 2 human-hours + 0 compute-hour. 2GB disk should be sufficient to store all the results]: In this experiment, we leverage results stored in previous executions of DeResistor and Geneva to select strategies that are effective for censorship evasion.

How to: We inspect collected strategies of DeResistor in `/generations/hall[final].txt` located in the folder corresponding to the DeResistor run. For real censors (e.g., China), we can evaluate the effectiveness of each strategy against the censor using Geneva startegy-testing Engine. More details about testing a strategy with Geneva engine is provided in *Execution*. You can also refer to Geneva documentation related to how to run a strategy in `https://geneva.readthedocs.io/en/latest/intro/gettingstarted.html#running-a-strategy`. For startegy generated against mock censors, We cannot evaluate them using Geneva engine. Instead we can manually inspects the logs of the most fit strategies and check whether the client finally had access to the forbidden server (evaded censorship). We note that, the most fit startegies are the ones that have the highest fitness values which can be negative in case we run DeResistor.

Preparation: To perform this experiment we need to generate appropriate results files using commands described in (E1). Similar to (E1), to test strategies against real censors (e.g. GFW), you need access to controlled vantage points in the desired country.

Execution:
- **Evaluate a strategy against real censors:** To evaluate a strategy that you selected, you first need to run Geneva engine to apply the strategy later on using:

  ```
  $ sudo /path/to/python_environment
  /bin/python engine.py --server-port 80
  --strategy "[your-strategy]" --log debug
  ```

  In a separate console (e.g. terminal), you can perform `curl` commands to attempt connections to a censored website. For instance, in china you can try:

  ```
  $ curl -L --no-keepalive --local-port
  [random-port-number] --connect-to
  ::23.185.0.2: 'http://hrw.org' -D -
  ```

  The port number has to be changed across runs to avoid Resisdual censorship performed by GFW (ths is discussed in the paper in §6.1 paragraph 3.
  To automate the strategy evaluation process, we provide a script in `/test.py` that evaluates a list of strategies 30 times and stores their success rate in `success_rate.txt`. Using this script, we can reproduce results in Table2.

Results:

.5 Notes on Reusability

[Optional] This section is meant to optionally share additional information on how to use your artifact beyond the research presented in your paper. In fact, a broader objective of an artifact evaluation is to help you make your research reusable by others.

You can include in this section any sort of instruction that you believe would help others re-use your artifact, like, for example, scaling down/up certain components of your artifact, working on different kinds of input or data-set, customizing the behavior replacing a specific module/algorithm, etc.