USENIX'23 Artifact Appendix: No more Reviewer #2: Subverting Automatic Paper-Reviewer Assignment using Adversarial Learning

Thorsten Eisenhofer*†, Erwin Quiring*†‡, Jonas Möller§, Doreen Riepel†, Thorsten Holz¶, Konrad Rieck§

†Ruhr University Bochum
‡International Computer Science Institute (ICSI) Berkeley
§Technische Universität Berlin
¶CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

The number of papers submitted to academic conferences is steadily rising in many scientific disciplines. To handle this growth, systems for automatic *paper-reviewer assignments* are increasingly used during the reviewing process. These systems use statistical topic models to characterize the content of submissions and automate the assignment to reviewers. In this paper, we show that this automation can be manipulated using adversarial learning. We propose an attack that adapts a given paper so that it misleads the assignment and selects its own reviewers. Our attack is based on a novel optimization strategy that alternates between the feature space and problem space to realize unobtrusive changes to the paper. To evaluate the feasibility of our attack, we simulate the paper-reviewer assignment of an actual security conference (IEEE S&P) with 165 reviewers on the program committee. Our results show that we can successfully select and remove reviewers without access to the assignment system. Moreover, we demonstrate that the manipulated papers remain plausible and are often indistinguishable from benign submissions.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

There are no expected risks or others ethical concerns when executing the artifact.

A.2.2 How to access

The code for the artifact is available on GitHub github.com/RUB-SysSec/adversarial-papers with commit hash `01fc915612c7ca72481b50ab7700dde1e0fa6188`.

*Shared first authorship

The main experiments require the following files

```
evaluation
|-- models
|   |-- overlap_0.70
|   |-- victim
|-- problemspace
|   |-- bibsources
|   |-- llms
|   |-- synonyms
|-- submissions
|   |-- oakland_22
|-- targets
    |-- budget-vs-transformer.json
    |-- featurespace-search.json
    |-- surrogates
        |-- surrogate_targets_4.json
```

Targets files and bib sources are included in the repository. Pre-trained models and required target submissions are provided at *blinded*. Due to licensing issues, we cannot make these submissions publicly available. We do, however, publish all of our crawling scripts, dummy examples, and pre-trained models. Refer to the repository for more information.

A.2.3 Hardware dependencies

The evaluation does not require special hardware. All experiments can be performed on a "regular" server using only CPUs. We performed the experiments on a server with 256 GB RAM and two Intel Xeon Gold 5320 CPUs.

A.2.4 Software dependencies

We provide a docker container that setups the required environment and which can be used to run the experiments.

A.2.5 Benchmarks

There are no benchmarks required to evaluate this artifact.

A.3 Set-up

A.3.1 Installation

For ease of use, we include a Dockerfile with all necessary tools to reproduce the results from the paper. It can be build via

```
git clone
↪ https://github.com/RUB-SysSec/adversarial-papers.git
↪ adversarial-papers
cd adversarial-papers; ./docker.sh build
```

After building the container, it is possible to spawn a shell with

```
./docker.sh shell
```

All containers get automatically deleted after the shell exits (cf. the `--rm` flag from `docker run`). To make the evaluation results both easily accessible and persistent, we map subdirectories of the evaluation folder `evaluation` from the host inside the container. To setup all paths correctly, it is therefore necessary to invoke the `docker.sh` in the base directory of the project.

A.3.2 Basic Test

After building the docker container, you can test your setup by running the following command

```
./docker.sh run "python3
↪ /root/adversarial-papers/src/attack.py --targets_file
↪ /root/adversarial-papers/evaluation/targets/test.json
↪ --format_level --workers 1 --trial_name basic-test"
```

This will start the attack for the target described in `evaluation/targets/test.json`. If everything is working properly, the attack should run for one iteration and immediately return successful. Results are stored in `evaluation/trials/basic-test`.

The main entry point for the attack is in the `src/attack.py` file. There are options provided to configure almost every aspect of the attack grouped into general, feature-space and problem-space specific configurations. See the documentation in the repository for further details.

When setting the number of workers to 1, the attack produces verbose output for debugging. For larger numbers, this output is *not* send to `stdout` but stored only as a log file in the respective result directory.

A.4 Evaluation workflow

The full evaluation consists of ten experiments, which requires about 6.5 CPU years to fully execute. In the following, we therefore describe only the subset of experiments we think are necessary to reproduce the major claims in the paper. Refer to the documentation in the repository for a complete description of all ten experiments include the hyperparameter search, and how to train your own models.

Each experiment is configured with a file describing all considered targets (`--targets_file`). These files are located at `evaluation/targets`. The scripts to re-generate these files are located at `scripts/targets`. Each target is optimized to run on a single-core and the experiments are therefore highly amneable for parallelization across CPU cores and machines. Note, that depending on the experiments more or less computer memory might be required (e.g., the black-box experiments require more memory per instance to store the surrogate models). Depending on the machine, this might limit the number of parallel executions. To get a good estimate, we will additionally report an approximated (!) maximal memory per instance (e.g., with 100 workers the experiment requires 100× the amount of this value).

Finally, for almost any experiment, it is possible to continuously check the current results which might allow to stop experiments early if the numbers have sufficiently converged (see the expected results for each experiment). Sending the interrupt signal (w/ CTRL+C) should usually stop all processes, but sometimes the scripts need a bit more persuasion. In this case, stopping the container proved to be an effective strategy (i.e., `docker kill <container-id>` with the container id either being autocomplete with pressing TAB or using `docker ps`).

A.4.1 Major Claims

In the paper, we investigate three major claims:

(C1): First, we show that the proposed attack is effective in crafting adversarial papers in a white-box setting. This is investigated with experiment (E1) described in the *feature-space search* paragraph in Section 4.1. The results of the experiment are reported inline in the text as well as Table 2.

(C2): Second, we demonstrate that the attack extends to different classes of transformations. This is described in the *all transformations* paragraph in Section 4.1 and is part of experiment (E2). The results of the experiment are reported in Figure 4.

(C3): Finally, we analyze if the attack remains viable in a black-box scenario as described in Section 4.2. We consider this in experiment (E3) and simulate the attack with varying numbers of surrogate models.

A.4.2 Experiments

(E1): *Feature-space search* [800 CPU hours + 31GB disk]: We start our evaluation by examining the feature-space search of our attack. For this experiment, we consider format-level transformations that can realize arbitrary changes. Other transformations are evaluated as part of experiment (E2).

The experiment can be executed with:

```
WORKERS=100
./docker.sh run "python3
↪ /root/adversarial-papers/src/attack.py
↪ --targets_file
↪ /root/adversarial-papers/evaluation/
targets/featurespace-search.json --reviewer_window 6
↪ --reviewer_offset 2 --no_successors 256
↪ --beam_width 4 --step 64
↪ --problem_space_block_features
↪ --feature_problem_switch 8 --format_level
↪ --workers ${WORKERS} --trial_name
↪ featurespace-search"
```

Per worker, roughly 850MB of memory are expected. Adjust the number of parallel executions accordingly. Raw results are stored in `evaluation/trials/featurespace-search` and can be analyzed with

```
./docker.sh run "python3 /root/adversarial-papers/
evaluation/scripts/00_featurespace_search.py"
```

Expected output (cf. Table 2 and inline in text)

```
FEATURE-SPACE SEARCH
[+] Overall success rate
    -> 99.67%

[+] Overall run-time
    -> median: 7m 12s

[+] Overall L1
    -> min   : 9
    -> max   : 22621

[+] Ratio between modifications and original content
    -> selection: 9.42%
    -> rejection: 13.37%

[+] Modifications per objective
        Selection Rejection Substitution
    L1         704      1032         2059
    Linf        17        43           62
```

(E2): *All transformations* [1200 CPU hours + 32GB disk]: In experiment (E1), we have focused on format-level transformations to realize manipulations. These transformations exploit intrinsics of the submission format, which effectively allows us to make arbitrary changes to a PDF file. In experiment (E2) we consider different classes of transformations as introduced in Section 3.2.

The experiment can be executed with:

```
WORKERS=100
./docker.sh run "python3
↪ /root/adversarial-papers/src/attack.py
↪ --targets_file
↪ /root/adversarial-papers/evaluation/
targets/budget-vs-transformer.json
↪ --problem_space_block_features --reviewer_window 6
↪ --reviewer_offset 2 --no_successors 256
↪ --beam_width 4 --step 64 --workers ${WORKERS}
↪ --trial_name budget-vs-transformer-1"
```

Per worker, roughly 2300MB of memory are expected. Adjust the number of parallel executions accordingly. Raw results are stored in `evaluation/trials/budget-vs-transformer` and can be analyzed with

```
./docker.sh run "python3 /root/adversarial-papers/
evaluation/scripts/04_all_transformations.py"
```

Expected output (cf. left part of Figure 4)

```
[+] Switches
    found no trials

[+] Budget
                    0.25   0.50   1.00   2.00   4.00
    Text      :   22.00  28.00  40.00  52.00  68.00
    + Encoding:   24.00  31.00  45.00  53.00  69.00
    + Format  :  100.00 100.00 100.00 100.00  99.00

[+] Saved plot @
↪ evaluation/plots/all-transformations.pdf
```

Note that the full plot in Figure 4 aggregates eight of such runs.

(E3): *Surrogates* [1000 CPU hours + 46GB disk]: In practice, an attacker typically does not have unrestricted access to the target system. We therefore also assume a black-box scenario and consider an adversary with only limited knowledge.

The experiment can be executed with:

```
WORKERS=50
./docker.sh run "python3
↪ /root/adversarial-papers/src/attack.py
↪ --targets_file
↪ /root/adversarial-papers/evaluation/targets/
surrogates/surrogate_targets_4.json --reviewer_window
↪ 2 --delta -0.16 --reviewer_offset 1
↪ --no_successors 128 --beam_width 4 --step 256
↪ --problem_space_block_features
↪ --feature_problem_switch 8 --format_level
↪ --workers ${WORKERS} --trial_name surrogates-4"
```

Per worker, roughly 2000MB of memory are expected. Adjust the number of parallel executions accordingly. Raw results are stored in `evaluation/trials/surrogates-4` and can be analyzed with

```
./docker.sh run "python3 /root/adversarial-papers/
evaluation/scripts/05_surrogates.py"
```

Expected output (cf. Figure 5 with ensemble size 4)

```
[+] Saved plot @ evaluation/plots/surrogates.pdf
```

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/`.