IvySyn: Automated Vulnerability Discovery in Deep Learning Frameworks

Neophytos Christou, Di Jin, and Vaggelis Atlidakis, *Brown University;*
Baishakhi Ray, *Columbia University;* Vasileios P. Kemerlis, *Brown University*

https://www.usenix.org/conference/usenixsecurity23/presentation/christou

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix

IvySyn: Automated Vulnerability Discovery in Deep Learning Frameworks

Neophytos Christou
Brown University

Di Jin
Brown University

Vaggelis Atlidakis
Brown University

Baishakhi Ray
Columbia University

Vasileios P. Kemerlis
Brown University

A Artifact Appendix

A.1 Abstract

This is the artifact appendix for the `IvySyn` fuzzing framework. It contains instructions about how to setup, run, and reproduce the results of `IvySyn`, along with information regarding system and resource requirements.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

`IvySyn` is a fuzzer for discovering security-critical bugs in Deep Learning (DL) frameworks. The list of APIs for which `IvySyn` has uncovered bugs, during our experiments, is available in the repository of the project. If, during the reproduction of any reported result, `IvySyn` produces PoVs for APIs not listed in project repository, contact the authors via HotCRP or follow the *Responsible Disclosure* steps in `README.md` to report the (newly discovered) vulnerabilities to the developers of the corresponding framework(s).

A.2.2 How to access

`IvySyn` is available at: `https://gitlab.com/brown-ssl/ivysyn/-/tree/4b3d26dda0ddea11282c2658e28090a738dfd6c7` (stable ref.)

A.2.3 Hardware dependencies

The provided Docker images are configured to use 4 CPUs and 16GB of RAM, but can also be set to use fewer (or more) resources, as needed.

A.2.4 Software dependencies

We provide a Docker image that builds and runs `IvySyn`, and hence Docker is required. Some scripts run outside Docker containers and were tested on Debian v11—but they are relatively simple and should work on any Linux distribution.

A.2.5 Benchmarks

All the data required for running our benchmarks are either included in the project repository or can be produced by our scripts during setup. Note that the prototype implementation of `IvySyn` fuzzes both CPU- and GPU-specific implementations of DL kernels. However, we are not able to provide access to machines with GPUs. Therefore, the benchmarks in the artifact fuzz only kernels with CPU implementations. This does not have any effect on the claims of the paper, other than a smaller number of instrumented and fuzzed kernels.

A.3 Setup

A.3.1 Installation

To setup `IvySyn`, simply invoke `docker/download-prebuilt-image.sh`. This script will download a pre-built Docker image of `IvySyn`. Alternatively, in order to build the `IvySyn` Docker image from scratch, invoke the script `docker/build-docker-image.sh`, under the root directory of the project repository. (Note that building the image from scratch requires ≈3.5 hours on a 16-core, 64GB RAM host; and the total size of the fully built image is ≈35GB.)

- To start the container, simply run:
  `docker/run-docker-image.sh`
  (This script is configured to run the container with access to 4 CPUs and 16GB of RAM.)

- To provide more/less resources to the container, use the optional `--memory` (e.g., `--memory 8g`) and `--cpus` (e.g., `--cpus 2`) arguments to the `run-docker-image.sh` script. If you increase the number of CPUs the container can use, you should also increase the amount of RAM, since more parallel jobs may require more memory.

- To get a shell on the container, run:
  `docker exec -it ivysyn-instance /bin/bash`

If you wish to run the experiments for comparing IvySyn with the two other fuzzers, namely Atheris and DocTer, an additional ≈39GB of storage is required for their corresponding Docker images (i.e., ≈26GB for Atheris and ≈13GB for DocTer). For more details, refer to A.4.1 and A.4.2.

A.3.2 Basic Test

For a quick smoke test, we recommend invoking the do-run.sh script, *inside the running Docker container of IvySyn*, and providing a small number of kernels to be fuzzed with the --nkernels argument. For example:

```
./do-run.sh --seed 1 --pytorch --nkernels 5
```

Once the script done, it should display a summary of the run, containing information about how to inspect the results.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): IvySyn *automatically* fuzzes DL frameworks and produces *Proofs of Vulnerability (PoVs)*—i.e., code snippets that trigger memory errors in low-level (C/C++) code of the respective framework via a high-level (Python) API.

(C2): IvySyn: (i) uncovers *more crashes* in the DL frameworks than the state-of-the-art Python fuzzer Atheris, and (ii) it does so *faster*. This is proven by the experiment described in Section 7.2 of our paper.

(C3): IvySyn produces *more PoVs per unit of time* than DocTer, yet another DL-framework fuzzer. This is proven by the experiment in Section 7.3 of our paper.

A.4.2 Experiments

In what follows, we provide instructions on how to setup, run, and interpret the results of our experiments. All compute-time approximations assume you are using 16GB of RAM and 4 CPUs to run the Docker containers. For additional details, see README.md at the root of the project repository.

(E1): *[Up to 58 compute-hours – Up to 28 compute-hours with suggested configuration + Up to 2GB disk]: Run IvySyn on a selected framework and produce PoVs.*
Preparation: Run and connect to the provided (or custom-built) Docker image.
Execution: To perform a full run of IvySyn, invoke the do-run.sh script, *inside the running Docker container of IvySyn*, by providing an integer as the RNG seed and either --tensorflow or --pytorch to choose the framework to be fuzzed, as follows:

```
./do-run.sh --seed 123 --tensorflow
```

However, note that a full run will require ≈45 hours for PyTorch and ≈12 hours for TensorFlow. We advise restricting the amount of kernels that will be fuzzed, by providing the extra --nkernels argument. Furthermore, we suggest fuzzing 300 kernels for each framework. This will require ≈17 hours for PyTorch and ≈9 hours for TensorFlow, while still producing PoVs. Running on TensorFlow will require an additional 2 hours for the first invocation of the script, in order to compile the C++ developer-provided TensorFlow tests, which are also used by IvySyn. Execute IvySyn as follows:

```
./do-run.sh --seed <rng seed> --tensorflow
--nkernels 300
```

Results: Once fuzzing is done, IvySyn will produce PoVs under results/<framework>/ivysyn_povs (where <framework> is pytorch or tensorflow). To manually run and reproduce the PoVs, do the following (in the IvySyn Docker container):

1. Activate the environment of the pip-installed version of the corresponding framework. In the case of PyTorch, run: source /home/ivyuser/ivysyn/venv/anaconda3/bin/activate; conda activate pytorch-1.11-orig. For TensorFlow, run: source /home/ivyuser/ivysyn/venv/tensorflow-2.6-orig/bin/activate.
2. Run the PoVs produced under /home/ivyuser/ivysyn/results/<framework>/ivysyn_povs using python3 <pov.py> (where <pov.py> is the filename of the selected PoV).

(E2): *[Atheris-comparison] [Up to 400 compute-hours – Up to 26 compute-hours with suggested configuration + Up to 40GB disk space]: Run IvySyn and a selected variant of the Atheris fuzzer, and compare their efficiency at uncovering crashing inputs.*
Preparation: We provide a separate Docker image that sets-up the two variants of Atheris, namely Atheris+ and Atheris++, which are described in Section 7.1 of our paper. Similarly to the IvySyn image, you can build this image from scratch, by running:

```
comparisons/atheris_comp/docker_env/docker
/build-docker-image.sh
```

or download a pre-built version of the image by running:

```
comparisons/atheris_comp/docker_env/docker
/download-prebuilt-image.sh
```

Execution: To perform the experiment that compares IvySyn to Atheris, invoke the compare-fuzzers.sh script *outside the IvySyn container*.

You need to specify an RNG seed, the framework to fuzz (either `--tensorflow` or `--pytorch`), the `Atheris` variant (`--atheris1`, which corresponds to `Atheris+`; or `--atheris2`, which corresponds to `Atheris++`), and whether you want to limit the number of kernels to be fuzzed using the `--nkernels` argument. For example:

```
./compare-fuzzers.sh --tensorflow
--atheris2 --seed 123 --nkernels 50
```

We suggest running at least 50 kernels to get a decent approximation of the overall results.

The script above will:

1. Instrument the subset of 308 TensorFlow and 283 PyTorch kernels we performed our experiment on (depending on the chosen framework) and recompile the target framework.
2. Execute `IvySyn` on the target kernels, limiting the fuzzed kernels if `--nkernels` was specified.
3. Run the selected `Atheris` variant on the same subset of fuzzed kernels.
4. Output a summary of the results.

Results: The `compare-fuzzers.sh` script will display a summary of the results. Specifically, a new directory will be created at `results/<framework>/atheris_comp/` (where `<framework>` is `pytorch` or `tensorflow`), containing the following:

- `results.csv`: start/end timestamp, as well as whether a crash was found, for each API (CSV file).
- `total_time.txt`: total time of the run (text file).
- `fuzzer_logs_dir.txt`: name of the directory in the Docker container with the raw logs produced by the `IvySyn` fuzzer (text file).

The corresponding CSV that contains similar entries for the `Atheris` run can be found at `comparisons/atheris_comp_fuzzed_<framework>.csv` (where `<framework>` is either `pytorch` or `tensorflow`). The raw `Atheris` logs can be found in the `Atheris` Docker container at `/home/ivyuser/ivysyn-atheris/fuzzer_output`. To connect to the `Atheris` Docker container, in order to manually inspect the logs, run `docker exec -it atheris-instance /bin/bash`.

(E3): *[DocTer-comparison] [Up to 34 compute-hours – Up to 26 compute-hours with suggested configuration + Up to 15GB disk space]: Run `IvySyn` and `DocTer` and compare their effectiveness at producing PoVs.*

Preparation: We provide a separate Docker image that sets-up `DocTer`. Similarly to the `IvySyn` image, you can either build it from scratch, by running:

```
comparisons/docter_comp/docker_env/docker
/build-docker-image.sh
```

or download a pre-built version of the image, by running:

```
comparisons/docter_comp/docker_env/docker
/download-prebuilt-image.sh
```

Execution: To perform the experiment that compares `IvySyn` to `DocTer`, invoke the `compare-fuzzers.sh` script *outside the IvySyn container*. You need to specify an RNG seed, the framework to fuzz (either `--tensorflow` or `--pytorch`), the `--docter` flag, and whether you want to limit the number of kernels to be fuzzed using the `--nkernels` argument. For example:

```
./compare-fuzzers.sh --tensorflow --docter
--seed 123 --nkernels 50
```

We suggest running at least 50 kernels to get a decent approximation of the overall results.

The script above will:

1. Instrument the subset of 125 TensorFlow and 105 PyTorch kernels we performed our experiment on (depending on the chosen framework) and recompile the target framework.
2. Execute `IvySyn` on the target kernels, limiting the fuzzed kernels if `--nkernels` was specified.
3. Run `DocTer` on the same subset of fuzzed kernels.
4. Output a summary of the results.

Results: The `compare-fuzzers.sh` script will display a summary of the results. A new directory with the `IvySyn` results will be created at `results/<framework>/docter_comp/` (where `<framework>` is `pytorch` or `tensorflow`), and will contain files similar to the ones mentioned in the Atheris experiment (i.e., `results.csv`, `total_time.txt`, and `fuzzer_logs_dir.txt`).
The corresponding CSV that contains similar entries for the `DocTer` run can be found at `comparisons/docter_comp_fuzzed_<framework>.csv` (where `<framework>` is `pytorch` or `tensorflow`). The raw `DocTer` logs can be found in the `DocTer` Docker container at `/home/workdir/<framework>`. To connect to the `DocTer` Docker container, in order to manually inspect the logs, run `docker exec -it docter-instance /bin/bash`.

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/.