McFIL: Model Counting Functionality-Inherent Leakage

Maximilian Zinkus, Yinzhi Cao, and Matthew D. Green, *Johns Hopkins University*

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

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: McFIL: Model Counting Functionality-Inherent Leakage

Maximilian Zinkus
Johns Hopkins University
zinkus@cs.jhu.edu

Yinzhi Cao
Johns Hopkins University
yzcao@cs.jhu.edu

Matthew D. Green
Johns Hopkins University
mgreen@cs.jhu.edu

A Artifact Appendix

A.1 Abstract

Our Artifact submission encapsulates the version of our tool, McFIL, which was used in the evaluation of the work. We provide the source code in a dedicated GitHub repository along with archived relevant dependencies. As configured, this artifact is prepared to reproduce our evaluation results in an offline setting rather than evaluate any online secure protocols. Interested parties are welcomed to independently install our software and evaluate it at their leisure, and to submit feedback, issues, and/or pull requests to the open source project.

A.2 Description & Requirements

Our software tool is intended to perform an iterative analysis of a target functionality. At each iteration, the tool gathers available information (constraints within a SAT solver), generates new constraint systems based on a randomized sampling algorithm, solves these SAT problems, and discovers an approximately greedy-optimal result. It then tests this result against an "Oracle," configured by default to be an offline instantiation of the target functionality as a test harness stand-in for an online secure protocol. Our test harness generates a secret at the beginning of the loop (withholding it from the main algorithm), and then iteratively discovers (partially or completely) this secret by generating and executing queries to the Oracle.

We evaluated McFIL on an Intel Xeon CPU E5-2695 v4 at 2.10GHz (72 threads) with 500 GB memory. Our evaluation targeted relatively smaller benchmarks to enable randomized repetition, and therefore did not stress this system to its limits. Our evaluation used the following software dependencies:

- CryptoMinisat 5.8.0 https://github.com/msoos/cryptominisat/releases/tag/5.8.0
- ApproxMC 4.0.1 https://github.com/meelgroup/approxmc/releases/tag/4.0.1
- Z3 4.8.15 https://github.com/Z3Prover/z3/releases/tag/z3-4.8.15
- louvain-community@8cc5382d https://github.com/meelgroup/louvain-community
- arjun@407ea7f5 https://github.com/meelgroup/arjun
- Python 3.8

A.2.1 Security, privacy, and ethical concerns

None directly, as our artifact is configured to evaluate functionalities in an offline setting by default, requiring the user to configure their target functionality. McFIL can be used in an "online" setting to directly evaluate real-world secure protocols and attempt to maximize leakage. We acknowledge that this could potentially be used to exploit target protocols, however, as a community we move forward and publish these tools to improve understanding and defense with the assumption that attackers will independently arrive at optimal attacks in secret.

A.2.2 How to access

The source code can be accessed at our GitHub release URL: https://github.com/maxzinkus/McFIL-Release/releases/tag/release

A.2.3 Hardware dependencies

Please refer to Description & Requirements above. We believe that the artifact can be run on a lower-specification machine sufficiently well to observe functionality and perform limited experiments (listed in Experiments) which indicate our broader results without requiring them to be fully re-run (which would take many compute-hours).

A.2.4 Software dependencies

Please refer to Description & Requirements above. We have bundled these dependencies within a Docker image for easier evaluation.

A.2.5 Benchmarks

This largely does not apply to our work. However, the included `target_funcs` folder contains example target functionalities which we evaluate against in our paper, and so these could be considered benchmarks in a sense. These are automatically discovered and used when their names are passed as command-line arguments to our tool such as `python3 main.py millionaires` for `target_funcs/millionaires.py`.

A.3 Set-up

Generally, in order to prepare a system for use with our tool, two groups of dependencies must be installed. First, the external package dependencies listed in Description & Requirements, and second the `python3 pip` dependencies in the software's `requirements.txt`.

A.3.1 Installation

[Mandatory] Instructions to download and install dependencies as well as the main artifact. After these steps the evaluator should be able to run a simple functionality test.

1. Install the dependencies
    - python 3.8: `sudo apt install python3`
    - `sudo apt install build-essential cmake zlib1g-dev libboost-program-options-dev libsqlite3-dev libgmp3-dev`
    - louvain-community: clone the repository and follow build instructions
        (a) `cd louvain-community ; mkdir build ; cd build ; cmake .. ; make ; sudo make install`
    - z3: clone the repository and follow build instructions
        (a) `cd z3 ; python scripts/mk_make.py --python ; cd build ; make ; sudo make install ; pip install z3-solver`
    - cryptominisat: clone the repository and follow build instructions
        (a) `cd cryptominisat ; mkdir build ; cd build ; cmake .. ; make ; sudo make install ; sudo ldconfig`
    - arjun: clone the repository and follow build instructions
        (a) `cd arjun ; mkdir build ; cd build ; cmake .. ; make ; sudo make install`
    - approxmc: clone the repository and follow build instructions
        (a) `cd approxmc ; mkdir build ; cd build ; cmake .. ; make ; sudo make install`
2. Fetch the software and install python dependencies
    (a) clone or otherwise fetch the source of McFIL
    (b) create a virtual environment `python3 -m venv venv`
    (c) install python dependencies `source venv/bin/activate ; pip install -r requirements.txt`

A.3.2 Basic Test

In order to determine if the dependencies are installed and the environment configured, the following commands should work without error with the virtual environment active:

- `cryptominisat </dev/null`
- `approxmc </dev/null`
- `python3 -c 'import z3 ; z3.SolverFor'`

Then, McFIL can be used to evaluate functionalities in the `target_funcs` directory such as:

- `python3 main.py millionaires`
- `python3 main.py sugarbeets`

A.4 Notes on Reusability

McFIL is designed for use with functionalities of the user's choosing. A critical step in analyzing a novel functionality is accurately encoding it in the input format that McFIL expects. We recommend that users of our software use the existing `target_funcs` given target functionality examples as a basis (e.g. by copy-pasting them) to work from when defining new targets. McFIL requires that the target be implemented both "in the clear" (i.e. a correct python implementation of the function under test) and in a format the solver can understand. We provide `solver.py`, a support library which we hope makes encoding easier. All existing examples use this library and can be referred to for aid in its use.

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/.