Silent Bugs Matter: A Study of Compiler-Introduced Security Bugs

Jianhao Xu, *Nanjing University;* Kangjie Lu, *University of Minnesota;* Zhengjie Du, Zhu Ding, and Linke Li, *Nanjing University;* Qiushi Wu, *University of Minnesota;* Mathias Payer, *EPFL;* Bing Mao, *Nanjing University*

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-jianhao

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

USENIX'23 Artifact Appendix: Silent Bugs Matter: A Study of Compiler-Introduced Security Bugs

Jianhao Xu[1] Kangjie Lu[2] Zhengjie Du[1] Zhu Ding[1] Linke Li[1] Qiushi Wu[2] Mathias Payer[3] Bing Mao[1]

[1]State Key Laboratory for Novel Software Technology, Nanjing University
[2]University of Minnesota [3]EPFL

A Artifact Appendix

A.1 Abstract

This artifact provides a dataset of Compiler-Introduced Security Bugs (CISBs). Our dataset comprises various types of CISBs, which we manually identified from GCC and Clang Bugzilla reports and Linux git history. We organize these CISBs in a taxonomy based on their root causes, formation, and security impacts. Additionally, we include test cases and their triggering oracles for all the reproduced CISBs. Please note that the user study data cannot be shared due to ethical considerations. We have promised our participants that we will only share statistics of their data.

To validate the results of our paper, we also prepare scripts to obtain statistics on the bugs in our dataset, reproduce the evaluation of compiler mitigations based on our dataset, and obtain statistics on the targeted bugs in our dataset for automatic prevention works in a console.

The minimum required disk space for the program is approximately 20 GB. We have tested it on Ubuntu 20.04. The software prerequisite for the program is an operating system that can run Docker and supports Ubuntu 20.04 as a container image. The whole experiment takes about 3 human-hours and 60-70 compute-hours.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The included scripts should not pose a greater risk to evaluators than regular benign Python scripts when running them.

A.2.2 How to access

The aritifact and dataset can be got from Github `https://github.com/H0w1/CISB-dataset/tree/aac22565c96744a13f0786854b3257d64421acef`.

A.2.3 Hardware dependencies

To run our evaluation, a x64 machine with a network connection is required.

A.2.4 Software dependencies

To evaluate the artifact, you need an operating system that can run Docker and supports Ubuntu 20.04 as a container image.

A.2.5 Benchmarks

Please note that running one experiment requires SPEC CPU 2006, which is not provided as it is not a free software. Reviewers will need to obtain their own copy of SPEC CPU 2006 to run this experiment.

A.3 Set-up

We provide a Dockerfile that automatically downloads the dataset and evaluation materials, as well as installs all the necessary software requirements.

Instructions for downloading and using the Dockerfile can be found on the `README` page `https://github.com/H0w1/CISB-dataset#aritifact-setup`.

A.3.1 Installation

After installing the Docker container from the Dockerfile, the dependencies and main artifact will be automatically prepared.

A.3.2 Basic Test

A basic functionality test can be easily executed by running the Python script 'check-compiler.py' using the command `'python3 check-compiler.py'`. Upon successful execution, the script will output a list of notes indicating that each compiler used has been installed correctly. You can find this script in the main directory of the Git repository `https://raw.githubusercontent.com/H0w1/CISB-dataset/`

b122ac42ff52ecb59b94a319c4558b1275cc9166/check-compiler.py.

A.4 Evaluation workflow

The overall workflow comprises the following steps:

1. Obtain the Dockerfile;
2. Obtain the dataset and test scripts and install the necessary dependencies automatically using the Dockerfile;
3. Execute a Python script to check the statistics of CISBs in the dataset;
4. Execute scripts and review some CISB details in the dataset to evaluate the effectiveness of current mitigations and the performance overhead of current compiler mitigations;
5. Execute a Python script to obtain statistics on the percentage of CISBs in our dataset that can be targeted by automatic prevention works. Check the statistics by reviewing related CISBs.

A.4.1 Major Claims

(C1): *We identify a large set of different kinds of CISB in the real world.* This is proven by the CISBs in the dataset. The statistic of these bugs can be viewed by running the script 'statistic.py'.

(C2): *We investigate and show the risks of existing mitigations.* Specifically, (i) we show CISB prevention performed by programmers is risky, derived from real cases; (ii) we perform a comprehensive evaluation of existing mitigations provided by compilers, with our dataset. We can prove (i) by pointing to the existence of a few bugs in our dataset. As for (ii), we can provide the script (('statistic.py')) we used to re-run the analysis and generate Table 6 (an evaluation of the mitigations provided by the compiler.)

(C3): *The CISBs we studied have not been extensively studied before.* This is proven by the script ('statistic.py') to get the results shown in Table 7, which shows the statistics of CISBs that can theoretically be prevented by automatic prevention works.

A.4.2 Experiments

We provide a guide of the experiments in the Github repository. https://github.com/H0w1/CISB-dataset#aritifact-experiments

(E1): *[CISB statistics] [30 human-minutes + 1 compute-second]:* Execute the Python script to obtain the statistics of CISBs in our dataset. Check the dataset and script for mistakes. The result should be in line with the data in Figure 2 and Figure 3 of the paper.

(E2): *[Evaulation of mitigations] [30 human-minutes + 60-70 compute-hours]:* (i) Review a list of bugs where the prevention performed by programmers failed. This list can be obtained by executing a script. It takes one compute-second. (ii) Run a script to obtain statistics on the effectiveness of compiler mitigations. It takes about two compute-minutes. (iii) Run scripts to measure the overhead of different compiler prevention strategies using the SPEC CPU 2006 benchmark. It takes about 60 compute-hours. For (i), the expected result is those CISBs exist. For (ii) and (iii), the output results should be in line with the data shown in Table 6 of the paper.

(E3): *[Target bugs of automatic prevention works] [2 human-hours + 2 compute-minutes]:* (i) Execute the script to obtain the statistics of CISBs that can theoretically be prevented by automatic prevention works. (ii) Check the lists of CISBs we summarized and shown in the script. For (i), the result should be in line with the data in Figure 7 of the paper. For (ii), these bugs should be within the scope of the corresponding prevention work.

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/.