jGuard: Programming misuse-resilient APIs

Artifacts for reproducibility

October 12, 2022

1.1 Overview

Our artifact consists of the following resources:

1. A video explaining how to use jGuard, `https://youtu.be/tnQGMyMjZRA`. Please use this this video to test if jGuard works, it serves as a test for Kick-the-Tires.

2. A public, anonymous repository, `https://github.com/jguardnm86/sle22`. This repository contains the MPS implementation for jGuard described in our paper. It also contains annotated code for BouncyCastle and core JDK classes which serve as the basis for our evaluation.

3. A repository of our benchmarks with usage instructions for reproducing them, available under `https://we.tl/t-9rSNP1Io3f`.

4. For convenience, a prepared virtual machine (OVA file) and pre-compiled versions of our verified BouncyCastle implementation are available under `https://zenodo.org/record/5767812#.YbDYZ1Mo9MQ`.

1.2 Setup

The video linked in the previous section explains how to setup jGuard. This section gives an alternative textual description. While our implementation is available as a VM, we recommend checking out jGuard from source if possible.

1.2.1 From sources

Compiling jGuard requires version 2021.2 of JetBrains MPS[1], no additional dependencies are needed. Clone our repository (`https://github.com/jguardnm86/sle22`) and load this directory as an MPS project. In MPS, first build the `apiSL` language implementing jGuard as described in our paper. For this step, model checker warnings in MPS if any occur. Next, build the `apiSL.runtime` model. At this point, all jGuard features are ready to be used in that MPS instance.

[1] Available for download under `https://www.jetbrains.com/mps/download/previous.html`

1.2.2 Via the prebuilt VM

We have prepared a virtual machine with MPS and jGuard installed. To review jGuard this way, download the linked OVA file and import it into a suitable virtualization software (we were using VirtualBox version 6.1).

Launch in the VM. You should be taken to the desktop directly. If needed, login with the user and password `jguard`. Launch MPS from the side bar to open the project which has already been compiled.

1.3 Reproducing our evaluation

After launching MPS with our implementation (and building the language if not using the prepared VM), the `sandbox` and `BouncyCastle` modules can be used to reproduce our evaluation resultse for RQ1-a and RQ1-b, respectively.

1.3.1 RQ1-a (Expressing guards against frequent misuses)

The `documents/misusessummary.pdf` file in the linked GitHub repository gives an overview of the most frequently occuring misuses in the MUBench dataset. Further, it describes how we designed jGuard checks against these misuses.

As it is not possible to reliably replace core JDK classes on an unmodified VM, we rely on custom tests in the `test` package in the `apiSL.sandbox` solution. These tests construct instances of our verified classes and trigger a misuse and can be run independently. The verified classes will throw an exception when this misuse occurs.

1.3.2 RQ1-b (Expressing guards against crypto misuses)

You can use the `BouncyCastle` model in the MPS project to compile a verified implementation of the BouncyCastle JCA implementation.

We have used an incremental approach to verify BouncyCastle and only converted the classes needing modifications to express guards against misuses. To create a usable `jar` file from these files, replace the affected classes from an unmodified BouncyCastle build. For your convenience, we have prepared a BouncyCastle jar with our changes which can be used to run benchmarks or verify misuses.

For running CogniCrypt as a static analysis tool, we refer to the documentation of $\text{CogniCrypt}_{\text{SAST}}$. As a data source, we used cryptographic misuses in MUBench. We identified cryptographic misuses as those using at least one class in the `javax.crypto` package. A detailed overview of relevant misuses and the guards we have designed against them is also given in the second section of the `documents/misusessummary.pdf` file in our repository.

To demonstrate how the `verified_bc.jar` can be used, consider the `SimpleCryptoMisuse.java` file at the root of the linked repository. This simple program misuses a JCA API, which can be verified as follows:

1. Run `javac -cp benchmarks/poi-benchmark/bclibs/original_bc.jar SimpleCryptoMisuse.java` to compile the application.

2. Run it against the original BouncyCastle implementation: `java -cp benchmarks/poi-benchmark/bclibs/original_bc.jar:. SimpleCryptoMisuse`. No misuse is reported.

3. Run it against the verified BouncyCastle implementation: `java -cp benchmarks/poi-benchmark/bclibs/verified_bc.jar:. SimpleCryptoMisuse`, the misuse is reported.

Similar to this synthetic example, cryptographic misuses reported in MUBench can be reproduced by downloading the source mentioned in MUBench and running it against the verified BouncyCastle implementation.

1.3.3 RQ2 (Benchmarks)

Relevant sources for our benchmarks are available for download here. The archive also contains instructions for reproducing the results (`usage.md`).