Dynamically Generating Callback Summaries for Enhancing Static Analysis (Artifact)

Steven Arzt
Fraunhofer SIT | ATHENE – National Research Center for Applied Cybersecurity, Darmstadt, Germany

Marc Miltenberger
Fraunhofer SIT | ATHENE – National Research Center for Applied Cybersecurity, Darmstadt, Germany

Julius Näumann
TU Darmstadt | ATHENE – National Research Center for Applied Cybersecurity, Darmstadt, Germany

Abstract

Interprocedural static analyses require a complete and precise callgraph. Since third-party libraries are responsible for large portions of the code of an app, a substantial fraction of the effort in callgraph generation is therefore spent on the library code for each app. For analyses that are oblivious to the inner workings of a library and only require the user code to be processed, the library can be replaced with a summary that allows to reconstruct the callbacks from library code back to user code. To improve performance, we propose the automatic generation and use of precise pre-computed callgraph summaries for commonly used libraries. Reflective method calls within libraries and callback-driven APIs pose further challenges for generating precise callgraphs using static analysis. Pre-computed summaries can also help analyses avoid these challenges.

We present CGMiner, an approach for automatically generating callgraph models for library code. It dynamically observes sample apps that use one or more particular target libraries. As we show, CGMiner yields more than 94% of correct edges, whereas existing work only achieves around 33% correct edges. CGMiner avoids the high false positive rate of existing tools. We show that CGMiner integrated into FlowDroid uncovers 40 % more data flows than our baseline without callback summaries.

This artifact description describes how the artifacts can be build.

2012 ACM Subject Classification Software and its engineering → Dynamic analysis

Keywords and phrases dynamic analysis, callback detection, java, android

Digital Object Identifier 10.4230/DARTS.10.2.2

Funding This research work has been funded by the German Federal Ministry of Education and Research and the Hessian Min- istry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

Related Article Steven Arzt, Marc Miltenberger, and Julius Näumann, "Dynamically Generating Callback Summaries for Enhancing Static Analysis", in 38th European Conference on Object-Oriented Programming (ECOOP 2024), LIPIcs, Vol. 313, pp. 4:1–4:27, 2024.
`https://doi.org/10.4230/LIPIcs.ECOOP.2024.4`

Related Conference 38th European Conference on Object-Oriented Programming (ECOOP 2024), September 16–20, 2024, Vienna, Austria

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2024 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

The artifact allows to reproduce the data in the paper. It furter allows the user to conduct own analyses to identify callback edges in libraries that were not used in our experiments for the paper.

Note that reproducing all experimental results from the paper is computationally expensive due to the size of the experiments. It further requires setting up a proper dynamic analysis environment as well as the VUSC code scanner on which out analysis is built. We therefore provide an option to run downscaled version of the experiments that uses pre-computed callback summariees and conducts the downstream FlowDroid analysis based on these results. This smaller version of the experiments does not require VUSC and does not requires phones or emulators for the dynamic analysis.

2 Content

The artifact package includes:

- Code for the scaled-down evaluation. Can be found at *Code/ReplicateNumbers*
- Code to automatically generate artifical apps as mentioned in RQ1. Can be found at *Code/C-allbackGeneration*
- Code for the FlowDroid evaluation. Can be found at *Code/JobSubmissionTool*
- Code for test cases. Can be found at *Code/AndroidStudioProjects*
- Manually annotated results of the dynamic analysis (Summary).
  Can be found at *Results/Summaries/virtualedgesummaries-dcid-complete-annotated.xml*
- Manually annotated Edgeminer summaries.
  Can be found at *Results/Summaries/edgeminer-annotated.xml*
- Results for different RQs - partially precomputed to allow for the scaled-down evaluation to run even if the dynamic analysis was skipped. Can be found at *Results/*

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS).

4 Tested platforms

Ubuntu 24.04 LTS x64

5 License

The artifact is available under the LGPL2 license.

6 MD5 sum of the artifact

029ab0d1f181d7e1405e081027af44cb

7 Size of the artifact

346.4 MiB

A

A.1 Running the fast evaluation

For the fast evaluation, the approach is simple: Build the docker container and run it, e.g. using

```
docker run --rm -it $(docker build -q .) /bin/bash Code/RQ-FastEval.sh
```

The values in the text of the paper are custom LaTeX macros. These defintions for these macros, i.e., the mapping between the LaTeX command and the value that shall be inserted, is created automatically while running the fast evaluation. In other words, the artifact was also used to genetare the data in the paper. In addition to the macros in the continuous text, the data in "Table 1" and "Table 2" is also generated during the run.

The docker container runs the Code present in *Code/ReplicateNumbers/*, which prints out the values in standard output. You can search in the output for

- **** RQ 4: Prevalence of transfer functions ****
- **** RQ 5: Comparison with EdgeMiner ****
- **** RQ 7: FlowDroid client analysis ****

A.2 Complete Evaluation

For the computationally expensive full evaluation, the process is a bit more elaborate. We used a specialized Eclipse distribution (VUSC Development Environment (VDE)) which can be used to develop VUSC plug-ins. VDE can be obtained from us under either a free VUSC Academic License for non-profit research organizations or a commercial VUSC license for companies [1]. We provide the Eclipse project of our VUSC-Plugin used to generate Callbacks, which can only be build using this special Eclipse build environment, since it has dependencies to VUSC components in order to manage communication between the app and the computer and as an instrumentation framework [2].

A.2.1 Data Set

Note that the dataset is only needed for a complete evaluation including the dynamic analysis and FlowDroid. We used the dataset from AndroZoo [1] as a data source for our apps. Due to licensing restrictions, we cannot share the raw apps, but we provide md5 sums of the used apps in *Datasets/*-subset-md5.txt* corresponding to each research question that uses real-world apps. *Datasets/*-subset.txt* correspond to the files containing absolute file names to the actual apps and are used by the evaluation scripts for the large, expensive evaluation.

A.2.2 VUSC Setup

The *VUSC/server.conf.needsadaption* file can be used as a template for the VUSC configuration. Furthermore, the *VUSC* folder contains screenshots of the launch configuration we used in the VUSC Development Environment.

A.2.3 Verifying Claims

First, we show how to compute the numbers for RQ1, which is an time consuming process (Section A.2.3.1). We then elaborate on how to generate the numbers for RQ4 to RQ7 (Section A.2.3.2).

A.2.3.1 Baseline over the Dataset, obtain the summaries from RQ1 & generate RQ1 numbers [very expensive]

The paragraph "Baseline over the Dataset" uses the data set for RQ1 (`Datasets/RQ1-Callback-Generation-subset.txt` is needed). You will also need an Android SDK with `android.jar` files in the `platforms/platform-NUMBER` directory, with the Number standing for a SDK version. We have the Android SDK 34 installed.

[1] `https://www.sit.fraunhofer.de/en/offers/projekte/vusc/`

In order to regenerate the summaries, the dynamic analysis has to be performed using VUSC. VUSC is needed since we have dependencies on VUSC core plugins. The easiest approach to build research plugins against VUSC is to use the VUSC Development Environment (VDE). VDE already comess equipped with the VUSC core plugins. You only need to import the plugins for this research project from `Code/CallbackAnalysis`.

Note that VDE does not come with a Maven plugin. If you want to modify the `Replicate Numbers`, `JobSubmission` or `CallbackCodeGeneration` projects, which do not depend on VUSC and are meant to run outside the scanner, you can either use a different IDE of your choice, or simply install the Maven support into VDE using Eclipse's normal software installation features.

We have provided screenshots in the `VUSC` folder which show how the launch configuration should be configured. Adjust the `-Xmx` VM option on the arguments tab to provide a substantial maximum heap size. Note that the heap must always fit into your (free) system memory. For development, we used 16 GiB of memory. For a mass evaluation, that won't be sufficient. VUSC normally requires a minimum 32 GB of memory with 64 GB being recommended. Especially for large analyes with many apps, we recommend using as much memory as you can so that VUSC can parallelize scanning tasks.

The program arguments must include the path to the config file using `--configfile PATH-TO-SERVER-server.conf`. We have attached a sample configuration filee at `VUSC/server.conf.needsadaption`. Make sure to change the values in there according to the instructions inside. We cannot publicly redistribute the Android SDK due to their terms of use (`https://developer.android.com/studio/terms`), i.e., this needs to be downloaded and installed manually. The path to the Android SDK must be set specified in the `server.conf` file.

Once VUSC is started, the scanner is accessible via its web interface. You need to enter your license key (which you can find on your licensing documents or inside the customer portal) on `http://127.0.0.1:18080/settings/license-management` and set-up a device on the Device Management on `http://127.0.0.1:18080/settings/device-management`. Select `Android device` as device type. We really recommend to use a real device as we found emulators to be rather sluggish (and consume quite a bit of system memory themselves). The serial number can be found using `adb get-serialno` (use the adb that lies within the Android SDK referenced in the server.conf). Please do not use your private phone due to security concerns (we do not recommend installing random apps). For the test data set, we set the `Dynamic Analysis Runtime Seconds` to 15 seconds.

You can then start the analysis by either loading apps into the scanner using drap&drop on the web-based UI or using the `JobSubmission` tool that we provide for mass analysis.

Alternatively, you can set up a dedicated VUSC server with the research plugins installed. This makes sense if you want to run the experiments on a (headless) server machine that does not have VDE installed, or if you want seperate development from execution. For this approach, you first run the normal VUSC installer, equivalently to how a productive VUSC scanner installation would be set up. The `/etc/ci/server.conf` file should then be modified to use part of our config in `Content/VUSC/server.conf.needsadaption`. This is important, as it restricts the analyses to the research plugins. Otherwise, many unrelated analyses will run when we later submit the target apps as VUSC will conduct its normal security scan.

Afterwards, the research plugin must be installed into the scanner instance. The easiest way is to export the plugin from VDE is to export the entire OSGI feature. This can be done via the main menu in VDE: Select File/Export.../Deployable plug-ins and fragments and select the both reproduce.* plugins. Export the feature to a directory, for example `/tmp/abc`. Run `sudo ci installplugin file:///tmp/abc reproduce.callbackevaluation` in order install the plugin. Note that the VUSC server can also run on a different machine, you then only need to copy your directory to the other machine. Then run `ci restart` to restart VUSC. The new plugin will be picked up during this restart. You can then submit your analysis jobs as normal, i.e., either via the scanner's web UI or via the `JobSubmission` tool.

When "Device with ID 1 doesn't exist" appears in the log file during the analysis, the device has not been configured. Make sure your device is unlocked when using the dynamic analysis.

If you do not have the data set, you can use test cases from `Datasets/TestAPKs/` (the code for these is in `Code/AndroidStudioProjects/`). In the folder you can also find the ground truth as returned by the tool.

The code in `Code/CallbackAnalysis/Evaluation` is used to generate the numbers for RQ1.

A.2.3.2 RQ4, RQ5, RQ7: Generating all the numbers

In order to reproduce these numbers, there are two different ways. There is a fast approach that is shown in Section A.2.3.2. Alternatively, there is am evaluation from scratch that is computationally expensive (see Section A.2.3.2).

Recompute precomputed results (dataset needed). Note that the implementation checks whether precomputed results are available in `Results/RQ4-TransferFunctionsPrevalence`. We provide these precomputed results so that the raw dataset is not needed in order to compute the values from the paper. If you want to recompute those results, make sure that you have valid paths to all apps in the `Datasets/RQ4-PrevalenceTransferFunctions-subset.txt` file. Then delete the contents of the folder `Results/RQ4-TransferFunctionsPrevalence`.

Rerun FlowDroid evaluation (dataset needed, expensive). We already provide precomputed FlowDroid results, thus this step is optional.

In order to recompute the FlowDroid evaluation, the data set `Datasets/RQ7-FlowDroid-ClientAnalysis-subset.txt` is needed, which we cannot redistribute, but is obtainable from AndroZoo (Allix et al, ACM MSR, IEEE, 2016). When you are using the docker container, you can mount the path to the data set, in our example below we have the dataset in in `/opt/dataset` on the host computer. You also need to have the Android SDK installed, which must have at least one `android.jar` in the `platforms` directory.

```
docker run --rm -v
path-to-android-sdk-linux/platforms:/opt/android-sdk-linux/platforms -v
/opt/dataset:/path/to/ -it $(docker build -q .) /bin/bash
```

In the resulting bash prompt, make sure that `ls /path/to/appinventor.ai_blasetaze.-ScottPilgrim.apk` returns a file. Then run `/bin/bash Code/RQ7-FlowDroidEval.sh`. The script will take days to complete. Since the script file explictly specifies 250 GB heap size, Java will terminate without results when you do not have enough RAM. When the script is complete, do not exit the bash as this would delete all results, but run the evaluation directly in the same docker container: `/bin/bash Code/RQ-FastEval.sh`

References

1 Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2901739.2903508.

2 Marc Miltenberger and Steven Arzt. Extensible and scalable architecture for hybrid analysis. In *Proceedings of the 12th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis*, pages 34–39, 2023.