Getting to Know You: Towards a Capability Model for Java

Ben Hermann[1] Michael Reif[2] Michael Eichberg[3] and Mira Mezini[4]

Abstract: Developing software from reusable libraries lets developers face a security dilemma: Either be efficient and reuse libraries as they are or inspect them, know about their resource usage, but possibly miss deadlines as reviews are a time consuming process. In this paper, we propose a novel capability inference mechanism for libraries written in Java. It uses a coarse-grained capability model for system resources that can be presented to developers. We found that the capability inference agrees by 86.81% on expectations towards capabilities that can be derived from project documentation. Moreover, our approach can find capabilities that cannot be discovered using project documentation. It is thus a helpful tool for developers mitigating the aforementioned dilemma.

1 Summary

The efficiency of software development largely depends on an ecosystem of reuse [Bo99, Gr93]. Numerous software libraries are available that solve various problems ranging from numerical computations to user interface creation. The safe use of these libraries is an exigence for the development of software that meets critical time-to-market constraints.

However, when including software libraries into their products software developers entrust the code in these libraries with the same security context as the application itself regardless of the need for this excessive endorsement. For instance, a system that makes use of a library of numerical functions also enables the library to use the filesystem or make network connections although the library does not need these capabilities. If the library contains malicious code it could make use of them. In commonly used languages like Java no effective mechanism to limit or isolate software libraries from the application code exists. So developers face a dilemma: Either trust the component and finish the project in time or be secure, review the library's source code and possibly miss deadlines.

We propose to consider this excessive assignment of authority as a violation of the *Principle of Least Privilege [SS75]*. The principle states that every program should operate under the least set of privilege necessary to complete its job. In order to alleviate the described dilemma, we introduce an effective mechanism in this paper to detect the actual permission need of software libraries written in Java.

Drawing inspiration from Android, we construct a capability model for Java. It includes basic, coarse-grained capabilities such as the authority to access the filesystem or to open a

[1] Technische Universität Darmstadt, Fachbereich Informatik Fachgebiet Softwaretechnik, Hochschulstraße 10, 64289 Darmstadt, hermann@cs.tu-darmstadt.de
[2] reif@cs.tu-darmstadt.de
[3] eichberg@cs.tu-darmstadt.de
[4] mezini@cs.tu-darmstadt.de

network socket. As Java programs by themselves cannot communicate with the operating system directly, any interaction with those capabilities has to happen through the use of the Java Native Interface (JNI). By tracking the calls backwards through the call graph, we produce a capability set for every method of the Java Class Library (JCL) and by the same mechanism towards methods of a library. We can thus effectively infer the necessary capabilities of a library using our approach. We can also infer the subset of these capabilities used by an application, as it may not use every functionality supplied by the library.

As the precision of our approach is directly depending on the precision of the algorithm used to calculate the call graph of the library, we took several measures to compute a reasonably precise call graph while not compromising the scalability of the algorithm too severely. We evaluated our approach by comparing our results against expectations derived from API documentation. We found that for 70 projects from the Qualitas Corpus [Te10], that we evaluated against, actual results exceeded expectations and produce a far more accurate footprint of the projects capability usage. Thereby, our approach helps developers to quickly make informed decisions on library reuse without the need for manual inspection of source code or documentation.

In our pursuit to mitigate the software developer's dilemma w.r.t. library reuse, we thus contribute the following in our paper:

- an algorithm to propagate capability labels backwards through a call graph,
- a labeling of native methods with their necessary capabilities to bootstrap the process,
- a collection of efficient analysis steps to aid the precision of common call-graph algorithms,
- an evaluation of the algorithm against extracted capability expectations from documentation.

We provide the implementation and all related data of our approach here:
`http://www.thewhitespace.de/projects/peaks/capmodel.html`

References

[Bo99] Boehm, Barry W: Managing Software Productivity and Reuse. IEEE Computer, 32(9):111–113, 1999.

[Gr93] Griss, Martin L: Software Reuse: From Library to Factory. IBM Systems Journal, 32(4):548–566, 1993.

[SS75] Saltzer, J.H.; Schroeder, M.D.: The protection of information in computer systems. Proceedings of the IEEE, 63(9):1278–1308, Sept 1975.

[Te10] Tempero, Ewan; Anslow, Craig; Dietrich, Jens; Han, Ted; Li, Jing; Lumpe, Markus; Melton, Hayden; Noble, James: Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In: 2010 Asia Pacific Software Engineering Conference (APSEC2010). pp. 336–345, December 2010.