Extraction of an Architectural Model for Least Privilege Analysis

Bernard Spitz, Grenoble Graduate School of Business, Grenoble, France, & KU Leuven, Leuven, Belgium
Riccardo Scandariato, IBBT-DistriNet, KU Leuven, Heverlee, Belgium
Wouter Joosen, IBBT-DistriNet, Department of Computer Science, KU Leuven, Heverlee, Belgium

ABSTRACT

This paper presents the design and implementation of a prototype tool for the extraction of the so-called Task Execution Model directly from the source code of a software system. The Task Execution Model is an essential building block for the analysis of the least privilege violations in a software architecture (presented in previous work). However, the trustworthiness of the analysis results relies on the correspondence between the analyzed model and the implementation of the system. Therefore, the tool presented here is a key ingredient to provide assurance that the analysis results are significant for the system at hand.

Keywords: Least Privilege, Reverse Engineering, Security, Software Architecture, Task Execution Model

INTRODUCTION

From an empirical investigation, least privilege (LP) showed to be the security principle of highest importance in secure software engineering (Buyens, 2007). The principle prescribes that every principal, i.e., user or computer process executing on behalf of a user, must be able to access only those computing resources and information that are necessary to complete its tasks (Saltzer, 1975). The incorrect enforcement of least privilege causes vulnerabilities related to elevation of privilege, which can be exploited by attackers to gain access to more resources than originally foreseen in the design. In turn, this leads to information leaks and abuse of the system.

In previous work, the authors introduced a technique to rigorously verify that a software architecture adheres to the principle of least privilege (Scandariato, 2010). The technique identifies the violations of the least privilege principle by leveraging a specific architecture model called Task Execution Model (TEM). Inspired by business process diagrams, the TEM represents the software architecture as a collection of principals that interact by means of operation invocations in order to complete...
the intended tasks. The TEM is a precise representation of the behavior of a system that is tailored for the purpose of identifying LP violations. When a new system is being developed, an early LP analysis can be done before the system is implemented. In this case, the TEM can be constructed starting from the software architecture documentation, namely the component diagrams, the deployment diagrams, and the sequence diagrams.

However, the development of the system might significantly deviate from the architectural design. This also happens after maintenance cycles, as often the documentation is not updated after changes are made. Therefore, the LP analysis needs to be reassessed after the system has been implemented or evolved. In this case, the TEM needs to be constructed starting from the source code, which is a tedious and error prone endeavor. The experience in four medium-sized projects revealed that undocumented behavior in the code of the four systems required the modification of the TEMs, which were initially built starting from the documentation only. The divergence between the documentation and the implementation is particularly detrimental from a security perspective. In fact, the analysis results would have been erroneous if the TEMs would have not been corrected. In our experience, the macro-structures of the design, like components and sub-systems, are in general properly documented. The majority of the inconsistencies are to be found at the level of the invocations among the components. Typically, only the main interactions among components are properly documented and additional communication paths that emerge at later stage (e.g., because of implementation-level optimizations) are missing. The least privilege analysis focuses on the interaction among components and, hence, is particularly affected by these inconsistencies. Therefore, the trustworthiness of the least privilege analysis is at stake if the conformance of the TEM with the final system is not assured.

As its main contribution, this paper provides a solution to the problem. We present the design and implementation of a prototype tool for the assisted recovery of the Task Execution Model from the source code. The prototype is built on top of a commodity software architecture recovery platform, namely Bauhaus (Raza, 2006). The prototype requires minimal human input. Namely, the user has to provide the system’s macro-structures, which are often properly documented, as observed before). This paper also presents the validation of the prototype in the context of a medium-size software project. The project has been previously analyzed for least privilege violations by a third party expert and, in that context, a correct TEM has been manually built by using both the available documentation and the code. In this paper, we use the prototype to generate the TEM and compare the results.

The main value of this paper’s contribution is the creation of an end-to-end chain for analysis of least privilege in software architectures: from the source code to the TEM (this work) and from the TEM to identification of least privilege violations (Scandariato, 2010).

In the rest of the paper, we introduce the Bauhaus platform and give more background information on the least privilege analysis. Then, we present the TEM extraction tool and its validation. Finally, we cover the related work and present the concluding remarks.

BACKGROUND: THE BUAHAUS TOOL

The Bauhaus tool suite (Raza, 2006) is a software architecture recovery tool developed at the universities of Bremen and Stuttgart. From the source code, Bauhaus creates two internal representations with different kinds of granularity: the intermediate language (IML), which contains low-level information, and the Resource Flow Graph (RFG), which provides a more abstract, high-level structure of the system under analysis. The latter format is used in this work to create a model that can be used to analyze least privilege violations.

The IML representation is populated using language specific parsers and allows low-level inspection of the source code such as code
Round-trip Engineering UML Class Models and Java Models: A Real-world Use Case for Bidirectional Transformations with QVT-R
[www.igi-global.com/article/round-trip-engineering-uml-class-models-and-java-models/170520?camid=4v1a](www.igi-global.com/article/round-trip-engineering-uml-class-models-and-java-models/170520?camid=4v1a)

Methods for Statistical and Visual Comparison of Imputation Methods for Missing Data in Software Cost Estimation
[www.igi-global.com/chapter/methods-statistical-visual-comparison-imputation/51974?camid=4v1a](www.igi-global.com/chapter/methods-statistical-visual-comparison-imputation/51974?camid=4v1a)