A verification approach to applied system security

Achim D. Brucker¹, Burkhart Wolff²

¹ Information Security, ETH Zürich, ETH-Zentrum, 8092 Zürich, Switzerland
e-mail: brucker@inf.ethz.ch
² Universität Freiburg, George-Köhler-Allee 52, 79110 Freiburg, Germany
e-mail: wolff@informatik.uni-freiburg.de

Abstract. We present a method for the security analysis of realistic models over off-the-shelf systems and their configuration by formal, machine-checked proofs. The presentation follows a large case study based on a formal security analysis of a CVS-Server architecture.

The analysis is based on an abstract architecture (enforcing a role-based access control), which is refined to an implementation architecture (based on the usual discretionary access control provided by the POSIX environment). Both architectures serve as a skeleton to formulate access control and confidentiality properties.

Both the abstract and the implementation architecture are specified in the language Z. Based on a logical embedding of Z into Isabelle/HOL, we provide formal, machine-checked proofs for consistency properties of the specification, for the correctness of the refinement, and for security properties.

Keywords: Verification – Security – Refinement – POSIX – Z

1 Introduction

These days, the Concurrent Versions System (CVS) is a widely used tool for version management in many industrial software development projects and plays a key role in open source projects usually carried out by highly distributed teams [3, 4]. (See http://www.cvshome.org.) CVS provides a central database (the repository) and means to synchronize local modifications of partial copies (the working copies) with the repository. The repository can be accessed via a network; this requires a security architecture establishing authentication, access control, and nonrepudiation. A further complication of the CVS security architecture stems from the fact that the administration of authentication and access control is done via CVS itself, i.e., the authentication table is accessed and modified via standard CVS operations.

This work emerged from our own experiences with setting up a CVS-Server for more than 80 users worldwide. Besides overcoming a number of security problems (see, e.g., http://www.cvshome.org/dev/security9706.html), we had to develop an improved CVS-Server configuration described in [1] meeting two system design requirements: first, we had to provide a configuration of a CVS-Server that enforces a role-based access control [13]; second, we had to develop an “open CVS-Server architecture,” where the repository is part of the shared filesystem of a local network and the server is a regular process on a machine in this network. While such an architecture has a number of advantages, the correctness and trustworthiness of the security mechanisms become a major concern. Thus, we decided to apply formal modeling and analysis techniques to meet the challenge.

In this paper, we present the method we developed for analyzing the security problems of complex systems such as the CVS-Server and its configuration. As a result, we provide the following contributions:

1. A modeling technique that we call architectural modeling, which has an abstraction level in between the usual behavioral modeling used in protocol analysis and code verification;
2. A technique to use system architecture models for defining security requirements;
3. The presentation of the mapping from security requirements to concrete security technologies as a data-refinement problem;
4. Mechanized proof techniques for refinements and security properties over system transitions; and
5. Reusable models for widely used security technologies.

In particular, we provide means to model a certain type of security policies and show how security analysis can be performed not only on the abstract but also on the concrete level.

The paper is organized as follows. After introducing some background material, e.g., CVS, our chosen specification formalism Z, and the architectural modeling style, we present the model of the abstract system architecture. We proceed with the model of the POSIX filesystem as an infrastructure for the implementation architecture and present the implementation architecture itself. Then we describe the refinement relation between the system architecture and the implementation architecture, and the analysis of security properties at the different layers based on formal proofs in an interactive theorem prover.

2 Background

2.1 The CVS operations

For the purpose of this paper, it is sufficient to mention only the most common CVS commands (initiated by the client). These are: login for client authenticating, add for registering files or directories for version control, commit for transferring local changes to the repository, and update for incorporating changes from the repository (e.g., fetching the latest version from the repository) into the working copy. Additionally, CVS provides functionality for accessing the history, for branching, for logging information (which is beyond the scope of this paper), and it provides a mechanism for conflict resolution (e.g., merging the different versions), which is only modeled as an abstract operation. Further, in order to facilitate both the refinement and the security analysis, we will include in our CVS model a operation that is, strictly speaking, not part of CVS but part of the operating system: the operation modify. This operation models changes of the working copy, e.g., by editing a file.

2.2 Z and Isabelle/HOL-Z

As our specification formalism, we chose Z [16] for the following reasons: first, Z fits our modeling problem since the complex states of our components suggest using a formalism with rich theories for data structures. Second, the syntax and semantics of Z are specified in an ISO standard; for future standardization efforts of operating system libraries (e.g., similar to the POSIX [17] model in Sect. 3.3.2), Z is therefore a likely candidate. Third, Z comes with a data-refinement notion [16, p. 136], which provides a correctness notion of the underlying “security technology mapping” between the two architectures and a means to compute the proof obligations. We assume a rough familiarity with Z (the interested reader is referred to excellent textbooks on Z such as [16, 18]).

As our modeling and theorem-proving environment we chose Isabelle/HOL-Z [2], an integrated documentation, type-checking, and theorem-proving environment for Z specifications built on top of Isabelle/HOL. Isabelle [9] is a generic theorem prover, i.e., new object logics can be introduced by specifying their syntax and inference rules. Isabelle/HOL is an instance of Isabelle with Church’s higher-order logic (HOL) [7], a classical logic with equality. Isabelle/HOL-Z is a conservative embedding of Z into HOL (which is semantically isomorphic to Z). As a result, Isabelle/HOL-Z combines up-to-date theorem-proving technology with a widespread, standardized specification formalism and powerful documentation facilities.

2.3 Architectural modeling

As a means to identify conceptual entities of the problem domain and to structure the overall specification, we found it useful to describe the architecture of the system on several abstraction layers. Following Garlan and Shaw’s approach [6, 15], architectures are composed of components (such as clients, servers, or stores like the filesystem) and connectors (like channels, shared variables, etc). In this terminology it is straightforward to make the mentioned architectures more precise (as implementation architecture, we present the intended “open server architecture,” see Fig. 1). We assume for each operation (such as add) a shared variable as connector that keeps all necessary information that goes to and from the components. This paves the way to formalize this architecture by describing the transition relation of the combined system by the parallel composition of the local transition relations of the components synchronized over the corresponding shared variable. Since such transition relations can be represented in Z by operation schemas, we can thus define, for example:

\[ \text{CVS}_{\text{add}} = \text{Client}_{\text{add}} \land \text{Server}_{\text{add}} \land \text{add}_{\text{shared variable}} \]

where \( \land \) is the schema-conjunction and \( \\text{\textbackslash} \) the hiding operator (i.e., an existential quantifier). Throughout this paper we will only present combined operation schemas and model properties over the transitive closure of their transition relations.

2.4 Architecture refinement

When analyzing security architectures one can separate an abstract security architecture (Sect. 3.2), which is