Chapter 13
JavaSPI: A Framework for Security Protocol Implementation

Matteo Avalle
Politecnico di Torino, Italy

Alfredo Pironti
INRIA, France

Davide Pozza
Teoresi Group, Italy

Riccardo Sisto
Politecnico di Torino, Italy

ABSTRACT

This paper presents JavaSPI, a "model-driven" development framework that allows the user to reliably develop security protocol implementations in Java, starting from abstract models that can be verified formally. The main novelty of this approach stands in the use of Java as both a modeling language and the implementation language. The JavaSPI framework is validated by implementing a scenario of the SSL protocol. The JavaSPI implementation can successfully interoperate with OpenSSL, and has comparable execution time with the standard Java JSSE library.

1. INTRODUCTION

Security protocols are distributed algorithms that run over untrusted networks with the aim of achieving security goals, such as mutual authentication of two protocol parties. In order to achieve such goals, security protocols typically use cryptography.

It is well known that despite their apparent simplicity it is quite difficult to design security protocols right, and it may be quite difficult to find out all the subtle flaws that affect a given protocol logic. Research on this topic has led to the development of specialized formal methods that can be used to rigorously reason about protocol logic and to prove that it does really achieve its

DOI: 10.4018/978-1-4666-2482-5.ch013

intended goals under certain assumptions (e.g., Blanchet, 2009).

One problem that remains with this solution is the gap that exists between the abstract protocol model that is formally analyzed and its concrete implementation written in a programming language. The latter may be quite different from the former, thus breaking the validity of the formal verification when the final implementation is considered.

In order to solve this problem two approaches have been proposed. On one hand, model extraction techniques (e.g., O'Shea, 2008; Bhargavan, Fournet, Gordon, & Tse, 2008; Backes, Maffei, & Unruh, 2010; Chaki & Datta, 2009), automatically extract an abstract protocol model that can be verified formally, starting from the code of a protocol implementation. On the other hand, code generation model-driven techniques (e.g., Pironti & Sisto, 2007; Kiyomoto, Ota, & Tanaka, 2008; Almeida, Bangerter, Barbosa, Krenn, Sadeghi, & Schneider, 2010; Bhargavan, Corin, Deniélou, Fournet, & Leifer, 2009; Balser, Reif, Schellhorn, Stenzel, & Thums, 2000; Song, Perrig, & Phan, 2001), automatically generate a protocol implementation, starting from a formally verified abstract model. In either case, if the automatic transformation is formally guaranteed to be sound, it is possible to extend the results of formal verification done on the abstract protocol model to the corresponding implementation code.

Model-driven development (MDD) offers the advantage of hiding the complexity of a full implementation during the design phase, because the developer needs only focus on a simplified abstract model. Moreover, since the implementation code is automatically generated, it is possible to make it immune from some low-level programming errors, such as memory leakages, that could make the program vulnerable in some cases but that are not represented in abstract models.

However, MDD usually requires a high level of expertise, which limits its adoption, because formal languages used for abstract protocol models are generally not known by code developers, and quite different from common programming languages. For example, the user needs to know the formal spi calculus language in order to properly work with the Spi2Java framework (Pironti & Sisto, 2007).

Our motivation is to solve this problem and make MDD approaches more affordable. To achieve this, our contribution is the proposal of a new framework, based on Spi2Java, called *JavaSPI* (http://typhoon5.polito.it/javaspi/), where the abstract protocol model is itself an executable Java program.

This little but significant difference grants several different improvements over frameworks like Spi2Java:

- It is not necessary to learn a new completely different modeling language anymore (Java is also used as a modeling language);
- Standard Java Integrated Development Environments (ides), to which the programmer is already familiar, can be used to develop the security protocol model like it was a plain Java program, making full use of IDE features such as code completion, or live compilation;
- It is possible to debug (or *simulate*) the abstract model using the same debuggers Java programmers are used to;
- Thanks to Java annotations, information about low-level implementation choices and security properties can be neatly embedded into the abstract model.

The viability of the proposed approach is validated by a case study where interoperable client and server sides of a specific SSL scenario are implemented. The interoperability capabilities are demonstrated by running alternatively the client and the server against the OpenSSL 0.98o corresponding implementations, while the performances of the generated code are compared