OFMC Usage Overview

Sebastian Mödersheim
Information Security Group, Dep. of Computer Science, ETH Zurich, Switzerland
www.infsec.ethz.ch/~moedersheim

February 13, 2006

1 Introduction

The On-the-Fly Model-Checker OFMC has been developed since 2001, for the main part in the context of the two EU-Projects AVISS and AVISPA:

http://www.avispa-project.org

OFMC is based on the idea of the *lazy intruder* which is a symbolic, constraint-based approach to modeling an intruder. The most important new feature of this release is that the user can specify an algebraic theory on message terms, modulo which the analysis of the protocol is performed.

1.1 Input Languages

In AVISS and AVISPA, two languages where developed: the High-Level Protocol Specification Language HLPSL and the Intermediate Format IF. The HLPSL is meant for the users to specify protocols in a high-level way and the IF is meant as a more low-level language as input for several analysis tools like OFMC.

The translator from HLPSL to IF is not part of this release and must be downloaded separately from the AVISPA web-page. On this site, one can also find a large collection of examples, the AVISPA library. The IF translations of these files are included in this release.

Unfortunately, several new features of OFMC cannot be used with the current version of HLPSL:

- HLPSL supports only a built-in set of operators, so when declaring new operators and their algebraic properties (e.g. addition and multiplication), there is no way to specify them in HLPSL.
- Although HLPSL supports functions (in the latest version called `hash_func`), their use is intended as hash-functions only and they cannot be used to specify other kinds of functions like a 'public-key table function' which maps every agent to its public-key, for instance.[1]

[1] Although one may specify such a function, e.g. the HLPSL expression $\{message\}_{pk(A)}$ would be interpreted as a symmetric-key encryption, rather than a public-key encryption.

- HLPSL supports only *concrete sessions*, i.e. one must specify constants for the participants in each role (e.g. "alice starts a session with the intruder"). OFMC's *symbolic sessions* overcomes this restriction, as explained in section 3.

Therefore, we have also included a few examples in the IF language which cannot be specified in HLPSL.

1.2 Literature

We have also included in this release several papers and documents on the techniques behind OFMC:

- *OFMC: A symbolic model checker for security protocols.* This paper explains the basic model and the lazy intruder technique.
- *Algebraic Intruder Deductions.* This paper describes the class of algebraic theories that can now be handled by OFMC and how this is done.
- *User-Guide for Algebraic Intruder Deductions in OFMC* supplements the previous paper, explaining in detail the specification of algebraic theory files.
- *Deliverable 2.3: The Intermediate Format.* This document of the AVISPA project defines the IF language.

2 Using `OFMC`

Usage. OFMC is invoked by typing on the command-line

```
Ofmc <filename> [Options]
```

where `<filename>` is an IF file to be checked, and the following options are available:

`-theory <Theoryfile>` for specifying a custom algebraic theory — see *User-Guide for Algebraic Intruder Deductions in OFMC*, which describes also the default-theory that is used, when this option is not specified.

`-sessco` This option is explained in Section 4.

`-untyped` This option forces OFMC to ignore all types specified in an IF file. (This is equivalent to specifying no types at all in IF or to give every atom and variable the type `message`.)

`-d <DEPTH>` Using this option, one can specify a depth bound (a positive integer) for the search (the default being unbounded depth). In this case, OFMC uses a depth-first search (while the standard search strategy is a combination of breadth-first search and iterative deepening search).

`-p <PATH>` where `<PATH>` is a whitespace-separated list of positive integers. Using this option, one can "manually browse" the search tree, e.g.:

`-p` is the root node,

`-p 0` is the first (left-most) successor of the root node,

`-p 0 1` is the second successor (next to left-most) successor of the node obtained by `-p 0`.

An exception is raised if a path to a non-existing node is specified. This option can be specified in combination with all other options, however, it must be given as the last in the sequence of options.

3 Symbolic Sessions

A scenario is a finite number of sessions, where a session is an instantiation of all protocol roles agent names. The number of scenarios grows exponentially in the number of sessions (for finite scenarios, one can bound the number of agent names). In HLPSL, the user must specify each scenario manually and run it through the AVISPA tool, in order to check that the protocol has no attacks for a given number of sessions.

OFMC allows one to specify symbolic sessions which involves specifying just one scenario – with agent names replaced by *variables*. During the search for an attack, OFMC instantiates the agent names when necessary (lazily).

In the example file of the SRP protocol, we have two roles User and Host, and we have specified the following *symbolic scenario* (where $User$, $User2$, $Host$, and $Host2$ are of type *Agent*):

```
initial_state init1 :=

 ...
 state_srp_user(User,Host,...)
 state_srp_host(Host,...)
 state_srp_user(User2,Host2,...)
 state_srp_host(Host2,...)

 & User/=Host
 & Host/=i
 & User /=Host2
 & User2/=Host2
 & User2/=Host
 & Host2/=i
```

This specifies two sessions, one between $User$ and $Host$, and one between $User2$ and $Host2$. Observe that in this specification the $Host$ is independent from the name of the client, as the $Host$ will accept communication from everyone. Using the inequalities, we can further require that hosts and users are

disjoint (i.e. no user in one session can be a host in another session) and that all hosts are honest (unequal intruder). Such a specification is not mandatory, but some protocols require that certain roles can only be played by honest users, for instance.

4 Session Compilation

Session compilation is a feature that has been supported by OFMC for quite a while, but has lead to many questions, hence this extra section.

When specifying the option `-sessco`, OFMC will first perform a search with a passive intruder to check whether the honest agents can execute the protocol, and then give the intruder the knowledge of some "normal" session between honest agents. In the case certain steps cannot be executed by any honest agent, OFMC reports that the protocol is not executable and stops. If the executability check is successful, then the normal search with an active intruder is started, with the only difference that the intruder initially knows all the messages exchanged by the honest agents in the passive intruder phase.

This is helpful both for quickly finding replay attacks (rather than specifying lots of parallel sessions), but also for checking the sanity of the specification, namely that at least the 'legal execution' of the protocol is possible.

We recommend to check each protocol specification with option sessco first, to see if it is executable in the model of OFMC. If OFMC replies that it is not the case, then one should try to simulate the legal execution (the way the protocol was meant to be executed) using the path option. At some point, OFMC will not offer the next step of the legal execution, and that's the first point where probably a mistake in the rules has occurred. Indeed such debugging of specifications is not very convenient, and we hope to offer you soon a more improved option to inspect protocol specifications.

If one role can loop (i.e. remain in the same control state forever and make infinitely many steps), sessco is not possible (and OFMC aborts with an error message), but then the search in OFMC does not terminate either, unless you specify a depth bound for the search.

In any case, when it is not possible to establish executability of the protocol, one should still check the protocol without the sessco option. In this case, one can manually perform the session compilation by adding to the initial intruder knowledge the messages of one run of the protocol with the fresh data replaced by fresh constants.

We hope you enjoy using OFMC and find many interesting attacks.
Questions and comments welcome.