The AVISS Security Protocol Analysis Tool*

Alessandro Armando[1]**, David Basin[2], Mehdi Bouallagui[3], Yannick Chevalier[3], Luca Compagna[1], Sebastian Mödersheim[2], Michael Rusinowitch[3], Mathieu Turuani[3], Luca Viganò[2], and Laurent Vigneron[3]

[1] Mechanized Reasoning Group, DIST, Università di Genova, Viale Causa 13, 16145 Genova, Italy. Tel: +39 0103532216, Fax: +39 0103532948.
[2] Institut für Informatik, Universität Freiburg, Germany.
[3] LORIA-INRIA-Lorraine, Nancy, France

Abstract. We introduce AVISS, a tool for security protocol analysis that supports the integration of back-ends implementing different search techniques, allowing their systematic and quantitative comparison and paving the way to their effective interaction. As a significant example, we have implemented three back-ends, and then used the AVISS tool to analyze 30 of the 50 problems in the Clark-Jacob's protocol library.

Keywords: Security, State-space exploration, Model-Checking.
Category: B (Tool presentations)

1 Introduction

We describe the AVISS (Automated Verification of Infinite State Systems) tool for security protocol analysis, which supports the simple integration of different back-end search engines. As examples, we have implemented back-ends based on an on-the-fly model-checker, a model-checker based on constraint logic, and a SAT-based model-checker. Although each of these checkers can work independently, integrating them into a single tool allows a systematic and quantitative comparison of their relative strengths and paves the way to their effective interaction. As an initial experiment, we have used the tool to analyze and find flaws in 30 protocols reported as insecure (out of the 50 protocols of [2]) in [4].

The AVISS tool has a web-based graphical user-interface that aids protocol specification and allows one to select and configure different back-ends; the tool is available at the URL: `www.informatik.uni-freiburg.de/~softech/research/projects/aviss`.

2 The system

The AVISS tool supports automatic protocol analysis in the presence of an active intruder. As illustrated in Figure 1, the system consists of different, independent

* This work was supported by the FET Open Assessment Project IST-2000-26410, "AVISS: Automated Verification of Infinite State Systems".
** Contact author, `armando@dist.unige.it`

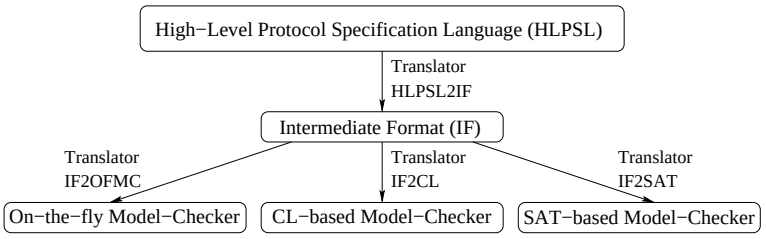

Fig. 1. The AVISS system architecture.

modules. Protocols are formulated in a high-level protocol specification language (HLPSL). A translator, HLPSL2IF, checks the executability of the protocol by means of a static analysis and then compiles the protocol and intruder activities into an intermediate format (IF) based on first-order multiset rewriting similar to CIL [3]. The IF unambiguously specifies an infinite state transition system. Afterwards, different translators are employed that translate the intermediate format into the input language of different analysis tools.

The input language HLPSL supports the declaration of protocols using standard "Alice&Bob" style notation indicating how messages are exchanged between principals [2]. Additionally, one specifies type information, initial knowledge of principals, possible intruder behavior (e.g. variants of the Dolev-Yao model [2]), and information about sessions instances (given explicitly or implicitly by declaring roles, with possibly several instances in parallel).[1] Security objectives (authentication, correspondence, secrecy, short-term secrecy) can also be declared. For example, the HLPSL specification of (the authentication part of) the well-known Needham-Schroeder Public Key (NSPK) protocol [2] is:

```
PROTOCOL NSPK;
Identifiers                          Messages
  A, B: user;                          1.  A -> B:  {A,Na}Kb
  Na, Nb: nonce;                       2.  B -> A:  {Na,Nb}Ka
  Ka, Kb: public_key;                  3.  A -> B:  {Nb}Kb
Intruder_knowledge I, a, b, ka, kb, ki;
Goal correspondence_between A B;
```

Ease of tool integration was an important design consideration for the AVISS tool. We currently have implemented three back-ends for performing complementary automated protocol analysis techniques.

On-the-fly model-checker (OF): The transition relation specified by the IF is unrolled starting from the initial state producing an infinite tree that is model-checked on-the-fly. We use Haskell, a compiled lazy functional programming language, to modularly specify the search space, reduction methods, heuristics, and procedures, generalizing the method described in [1].

[1] Note also that the tool handles several types of keys: symmetric (atomic or non-atomic), asymmetric (public), and arrays of asymmetric keys are supported.

Constraint-Logic-based model-checker (CL): The IF is translated into a first-order theory which is fed to the daTac prover [6]. The CL model-checker combines rewrite-based first-order theorem proving with constraint logic in order to handle properties such as associativity/commutativity of operators for representing sets of messages. Message exchanges and intruder activities are directly translated from the IF rewrite rules into clauses; searching for a flaw then amounts to searching for an incoherence in the resulting formula.

SAT-based model-checker (SAT): The SAT-based model-checker builds a propositional formula encoding a bounded unrolling of the transition relation specified by the IF, the initial state, and the set of states representing a violation of the security properties. The propositional formula is then fed to a state-of-the-art SAT solver (currently Chaff, SIM, and SATO are supported) and any model found by the solver is translated back into an attack, which is reported to the user.

Overall, there are other tools for protocol analysis providing similar features, e.g. [3, 7, 8]. To our knowledge, only CAPSL [3], with its intermediate language CIL, is designed to support multiple analysis techniques. There are however several important differences between CAPSL/CIL and the AVISS tool. First, the CAPSL translator does not generate attacker rules, whereas we are able to produce IF rules from the specification of the intruder behavior. Second, we test a more general notion of implementability (useful for e-commerce protocols such as non-repudiation protocols). CAPSL is unable to translate protocols where a principal receives a cipher, say $\{Na\}_K$, and later receives the key K and then uses Na in some message. In our case, the principal will store $\{Na\}_K$ and will decrypt it when he later receives the key. Finally, based on the available experiments, the AVISS tool and its back-ends are considerably more effective on the Clark-Jacob's library than CAPSL and its current connectors.

3 Experiments

We have run the current version of the AVISS tool successfully on 30 flawed protocols of the Clark-Jacob's library [2, 4]. The following table lists the performance of the three back-ends on a selection of these problems. For each protocol we give the kind of attack found and the time this required.[2] For the SAT model-checker we give a pair of values t_e/t_s, where t_e is the *encoding time*, i.e. the time spent to generate the propositional formula, and t_s is the *search time*, i.e. the time spent by the SAT solver to check the formula. The experiments show that the OF model-checker outperforms the other two. However, it is interesting to observe that the search time of the SAT model-checker is comparable to that of the OF model-checker. Moreover, small changes in the protocol require only the re-computation of a small part of the SAT encoding. Therefore the encoding time may have a smaller impact in practical situations than the figures in the table below seem to indicate. The poorer timings of the CL model-checker are

[2] Times have been obtained on a PC with a 1.4 GHz Processor and 512 MB of RAM. The SAT timings are obtained using the SIM solver [5].

balanced by the fact that it is based on an off-the-shelf prover (daTac) and it offers other advantages such as the simple integration of algebraic relations on message constructors (e.g. commutativity of encryptions in RSA).

Protocol Name	Kind of Attack	AVISS		
		OF	SAT	CL
Encrypted Key Exchange (EKE)	Parallel-session	0.1	14.8/0.4	145.6
ISO Symmetric Key One-Pass Unilateral Authentication	Replay	0.1	16.3/0.1	2.0
Kao-Chow Repeated Authentication	Replay	0.5	?	48.1
Needham-Schroeder with Conventional Keys	Replay	0.4	?	41.9
Needham-Schroeder Public Key (NSPK)	Man-in-the-middle	0.1	3/0.1	11.0
NSPK with Lowe's fix	Type flaw	0.1	-	32.2
Neumann-Stubblebine (initial part)	Type flaw	0.1	-	4.5
Otway-Rees Key Exchange	Type flaw	0.1	-	10.2

The IF contains type information that helps the SAT model-checker to generate compact encodings. However, when looking for type-flaw attacks, type information must be neglected and this makes the SAT model-checker unsuited for seeking this kind of attacks. As a consequence, the search for such attacks is not supported by the current SAT back-end. Since, on the other hand, both the OF and the CL checkers are largely insensitive to the type information, our experiments are obtained by running them against an untyped version of the IF directly obtained by the HLPSL2IF translator.

We have begun experimenting with larger e-commerce protocols and the first results are very promising. For example, we have been able to compile and verify the card-holder registration phase of SET protocol. We are also working on implementing back-ends for other verification techniques.

References

1. D. Basin. Lazy infinite-state analysis of security protocols. In *Secure Networking — CQRE'99*, LNCS 1740, pp. 30–42. Springer, 1999.
2. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL `http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`.
3. G. Denker and J. Millen. CAPSL Intermediate Language. In *Proc. of FMSP'99*. URL for CAPSL and CIL: `http://www.csl.sri.com/~millen/capsl/`.
4. B. Donovan, P. Norris, and G. Lowe, Analyzing a library of protocols using Casper and FDR. In *Proc. of FMSP'99*.
5. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proc. of IJCAR'01*, LNAI 2083, pp. 347–363. Springer, 2001.
6. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proc. of LPAR'00*, LNCS 1955, pp. 131–160. Springer, 2000.
7. G. Lowe. Casper: a compiler for the analysis of security protocols. *J. of Computer Security*, 6(1):53–84, 1998. URL for Casper: `http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/index.html`.
8. C. Meadows. The NRL protocol analyzer: An overview. *J. of Logic Programming*, 26(2):113–131, 1996. `http://chacs.nrl.navy.mil/projects/crypto.html`.