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, Italy.
[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 for their systematic and quantitative comparison and paving the way to their effective interaction. As a significant example, we have implemented three back-ends, and used the AVISS tool to analyze and find flaws in 36 protocols, including 31 problems in the Clark-Jacob's protocol library and a previously unreported flaw in the Denning-Sacco protocol.

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 example back-ends, we have implemented an on-the-fly model-checker, an analyser based on constraint logic, and a SAT-based model-checker. Although each of these back-ends can work independently, integrating them into a single tool allows for the systematic and quantitative comparison of their relative strengths, and paves the way for their effective interaction. As an initial experiment, we have used the tool to analyze and find flaws in 36 protocols, including a previously unknown flaw in the Denning-Sacco protocol and previously reported attacks (see [4]) to 31 protocols of [2].

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

2 The system

The AVISS tool supports automatic protocol analysis in the presence of an active intruder. As illustrated in Fig. 1, the system consists of different, independent modules. Protocols are formulated in a high-level protocol specification language

* This work was supported by the FET Open Assessment Project IST-2000-26410, “AVISS: Automated Verification of Infinite State Systems”.

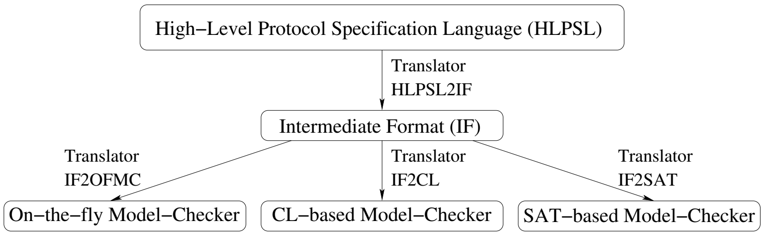

Fig. 1. The AVISS system architecture.

(HLPSL). The translator HLPSL2IF performs a static analysis to check the executability of the protocol (i.e. whether each principal has enough knowledge to compose the messages he is supposed to send), and then compiles the protocol and intruder activities into an intermediate format (IF) based on first-order multiset rewriting. The IF unambiguously specifies an infinite state transition system. Afterwards, different translators are employed that translate the IF into the input language of different analysis tools. The IF can be also generated in a typed variant (the untyped one is the default), which leads to smaller search spaces at the cost of abstracting away type-flaws (if any) from the protocol.

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), and information about session instances (given explicitly or implicitly by declaring roles, with possibly several instances in parallel).[4] Security objectives (authentication and 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 B authenticate A on Na;
```

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.

The On-the-fly model-checker (OFMC): The transition relation specified by the IF is unrolled starting from the initial state producing an infinite

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

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 of [1]. When an attack is found, it is reported to the user by means of the sequence of exchanged messages.

Constraint-Logic-based model-checker (CL): The IF is translated into a first-order theory which is input to the daTac prover [5]. The CL back-end 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.

The SAT-based Model-Checker (SATMC): 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.

There are other tools for protocol analysis providing similar features, e.g. [3, 6, 7]. 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 executability (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 published 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 successfully the AVISS tool to find flaws in 36 protocols, including 32 protocols from the Clark-Jacob's library [2, 4]. Of the 36 protocols analyzed, 31 were already reported to be insecure in [4] whereas for 1, namely the Denning Sacco protocol, no flaw was previously found. Table 1 lists the performance of the three back-ends on these problems.[5] Preliminary to the execution of the back-ends we generated both the untyped and the typed version of the IF specifications by means of the HLPSL2IF translator. The OFMC and the CL back-end were run against the untyped and the typed IF specifications of each protocol. The

[5] The protocols marked with a "*" are variants of protocols in [2] that we have additionally analyzed. Times are obtained on a PC with a 1.4GHz Pentium III processor and 512Mb of RAM. The SATMC timings are obtained using the Chaff solver [8].

kind of the attack found (if any) and the time spent by each back-end are given in the corresponding columns. For SATMC 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. Note that the analysis of the untyped and typed IF specifications may lead to the detection of different kinds of attacks. Since SATMC is not suited to analyze untyped IF specifications, we applied it to typed specifications only (and thus not on protocols suffering from type flaw attacks).

Table 1 allows us also to analyze and compare the performance of the individual back-ends. The OFMC model-checker performs uniformly well on all the protocols: most of the attacks are found in a fraction of a second, and detecting all the attacks requires a total time of less than one minute. The poorer timings of the CL back-end are 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). For SATMC, the experiments show that the time spent to generate the SAT formula largely dominates the time spent to check the satisfiability of the SAT instance. Nevertheless, in many cases the overall timing is not too far from that of OFMC and it is better than that of CL. It is also interesting to observe that in many cases the time spent by the SAT solver is smaller than the time spent by OFMC for the same protocol.

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 analyze the card-holder registration phase of SET protocol. Since our tool can only detect attacks (and for a correct protocol it only terminates when checking a finite number of sessions), we are also working on implementing back-ends that can find security proofs for correct protocols.

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. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proc. of LPAR'00*, LNCS 1955, pp. 131–160. Springer, 2000.
6. 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`.
7. 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`.
8. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proc. of DAC'01*. 2001.

Table 1. Performance of the AVISS tool back-ends over the testsuite

Protocol Name	Kind of Attack	AVISS		
		OFMC	CL	SATMC
ISO symm. key 1-pass unilateral auth.	Replay	0.0	2.0	0.2/0.0
ISO symm. key 2-pass mutual auth.	Replay	0.0	3.9	0.4/0.0
Andrew Secure RPC prot.	Type flaw Replay	0.0 0.1	4.3 32.7	NA 80.6/2.7
ISO CCF 1-pass unilateral auth.	Replay	0.0	2.2	0.2/0.0
ISO CCF 2-pass mutual auth.	Replay	0.0	4.6	0.5/0.0
Needham-Schroeder Conventional Key	Replay STS	0.3	63.4	29.3/0.4
Denning-Sacco (symmetric)	Type flaw	0.0	16.0	NA
Otway-Rees	Type flaw	0.0	10.7	NA
Yahalom with Lowe's alteration	Type flaw	0.0	44.1	NA
Woo-Lam Π_1	Type flaw	0.0	0.8	NA
Woo-Lam Π_2	Type flaw	0.0	0.8	NA
Woo-Lam Π_3	Type flaw	0.0	0.8	NA
Woo-Lam Π	PS	0.2	1075.0	3.3/0.0
Woo-Lam Mutual auth.	PS	0.3	245.6	1024.1/8.0
Needham-Schroeder Signature prot.	MITM	0.1	53.9	3.8/0.1
* *Neuman Stubblebine initial part*	Type flaw	0.0	6.2	NA
* *Neuman Stubblebine rep. part*	Replay STS	0.0	3.5	15.2/0.2
Neuman Stubblebine (complete)	Type flaw	0.0	46.8	NA
Kehne Langendorfer Schoenwalder (rep. part)	PS	0.2	199.4	MO/-
Kao Chow rep. auth., 1	Replay STS	0.5	76.8	16.3/0.2
Kao Chow rep. auth., 2	Replay STS	0.5	45.3	339.7/2.1
Kao Chow rep. auth., 3	Replay STS	0.5	50.1	1288.0/MO
ISO public key 1-pass unilateral auth.	Replay	0.0	4.2	0.3/0.0
ISO public key 2-pass mutual auth.	Replay	0.0	11.1	1.2/0.0
* *Needham-Schroeder Public Key*NSPK	MITM	0.0	12.9	1.8/0.1
NSPK with key server	MITM	1.1	TO	4.3/0.0
* NSPK with Lowe's fix	Type flaw	0.0	31.1	NA
SPLICE/AS auth. prot.	Replay	4.0	352.4	5.5/0.1
Hwang and Chen's modified SPLICE	MITM	0.0	13.1	NS
Denning Sacco Key Distr. with Public Key	MITM	0.5	936.9	NS
Shamir Rivest Adelman Three Pass prot.	Type flaw	0.0	0.7	NA
Encrypted Key Exchange	PS	0.1	240.8	75.4/1.8
Davis Swick Private Key Certificates, prot. 1	Type flaw Replay	0.1 1.2	106.2 TO	NA 1.4/0.0
Davis Swick Private Key Certificates, prot. 2	Type flaw Replay	0.2 0.9	348.5 TO	NA 2.7/0.0
Davis Swick Private Key Certificates, prot. 3	Replay	0.0	2.7	1.5/0.0
Davis Swick Private Key Certificates, prot. 4	Replay	0.0	36.0	8.2/0.1

Legenda: MITM: Man-in-the-Middle. PS: Parallel-Session. Replay STS: Replay attack based on a Short-Term Secret. NA: Not Attempted. NS: Not Supported. MO: Memory Out. TO: Time Out (> 1 hour).