The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols*

Tool Paper

Cas J.F. Cremers

Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland
cas.cremers@inf.ethz.ch

1 Introduction

With the rise of the Internet and other open networks, a large number of security protocols have been developed and deployed in order to provide secure communication. The analysis of such security protocols has turned out to be extremely difficult for humans, as witnessed by the fact that many protocols were found to be flawed after deployment. This has driven the research in formal analysis of security protocols. Unfortunately, there are no effective approaches yet for constructing correct and efficient protocols, and work on concise formal logics that might allow one to easily prove that a protocol is correct in a formal model, is still ongoing. The most effective approach so far has been automated falsification or verification of such protocols with state-of-the-art tools such as ProVerif [1] or the Avispa tools [2]. These tools have shown to be effective at finding attacks on protocols (Avispa) or establishing correctness of protocols (ProVerif).

In this paper we present a push-button tool, called *Scyther*, for the verification, the falsification, and the analysis of security protocols. Scyther can be freely downloaded, and provides a number of novel features not offered by other tools, as well as state-of-the-art performance. Novel features include the possibility of unbounded verification with guaranteed termination, analysis of infinite sets of traces in terms of patterns, and support for multi-protocol analysis.

Scyther is based on a pattern refinement algorithm, providing concise representations of (infinite) sets of traces. This allows the tool to assist in the analysis of classes of attacks and possible protocol behaviours, or to prove correctness for an unbounded number of protocol sessions. The tool has been successfully applied in both research and teaching.

2 The Scyther Tool

The tool provides a graphical user interface (Fig. 1), that complements the command-line and Python scripting interfaces. The GUI is aimed at users interested in verifying or understanding a protocol. The command-line and scripting interfaces facilitate the use of Scyther for large-scale protocol verification tests.

* This work was partially supported by the Hasler Foundation, ManCom project 2071.

A. Gupta and S. Malik (Eds.): CAV 2008, LNCS 5123, pp. 414–418, 2008.

Scyther combines a number of novel features with state-of-the art performance. First, Scyther is guaranteed to terminate whilst allowing to prove correctness of protocols for an unbounded number of sessions, and can optionally output the proof tree (by using the backend). In contrast to other unbounded verification tools, the tool provides useful results even in the case that no attack is found but also no unbounded correctness can be established. In such cases, the results from Scyther have a similar interpretation as bounded verification tools, stating that no attack exists within a certain bound.

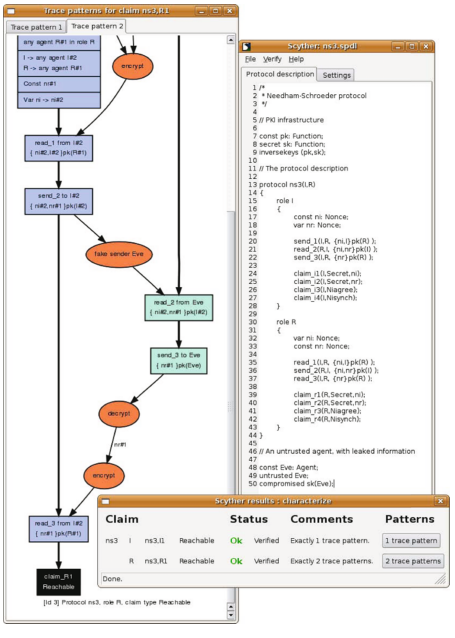

Fig. 1. The graphical user interface

Second, Scyther assists in protocol analysis by providing classes of protocol behaviour (or classes of attacks), as opposed to just single attack traces provided by other tools.

Third, Scyther facilitates so-called multi-protocol analysis. In such an analysis, the parallel composition of two (sub)protocols is analyzed, as in [3]. Traditionally, such an analysis has been infeasible for protocol tools because of state-space explosion. With the performance provided by the Scyther tool, multi-protocol analysis has become feasible, and can be performed by simply verifying the concatenation of multiple protocol description files.

Given the description of a protocol in the *spdl* language, which is based on the operational semantics found in [4], the tool can be used in three ways: to verify whether the security claims in the protocol description hold or not; to automatically generate appropriate security claims for a protocol and verify these; to analyze the protocol by performing *complete characterization*. We describe each of these modes below.

Verification of claims. The input language of Scyther allows for specification of security properties in terms of claim events, i.e., in a role specification one can claim that a certain value is confidential (secrecy) or certain properties should hold for the communication partners (authentication). Scyther can be used to verify these properties or falsify them.

Automatic claims. If the protocol specification contains no security claims, Scyther can automatically generate claims. At the end of each role, authentication claims are added, claiming that the supposed communication partners must have performed the protocol as expected. Similarly, secrecy claims are added for

all locally generated values (nonces) and variables. This augmented protocol description is then analyzed by Scyther as in the previous case. This enables users to quickly assess the properties of a protocol.

Characterization. For protocol analysis, each protocol role can be "characterized", which means that Scyther analyzes the protocol, and provides a finite representation of all traces that contain an execution of the protocol role. In most cases, this representation consists of a small number (1-5) of possible execution patterns. By manually inspecting these patterns, one can quickly gain insight in the potential problems with the protocol and modify it if necessary. For example, given the Needham-Schroeder protocol, Scyther determines that there are only two patterns for the responder role: one is the correct behaviour of the protocol, and the other is the well-known man-in-the-middle attack. Hence, there are no other possible ways of executing the responder role.

The algorithm developed for the Scyther tool extends on ideas described in [5], and the idea of analyzing protocols in terms of trace classes was published first in [6]. Scyther addresses the undecidability of the security problem by (1) significantly improving and extending the class pruning theorems from [5] and (2) introducing a parameter that limits the pattern size, ensuring termination. Even though the pattern depth size is limited, Scyther can perform unbounded verification of the majority of protocols, as each pattern represents an infinite class of traces. In practice, with protocols from libraries such as SPORE [7], Scyther is known to provide in about 80 percent of cases either unbounded verification or falsification, and in the other 20 percent provides bounded verification. Further details about the underlying methods are given in [8]. The Scyther tool is freely available for Windows, Linux, and Max OS X platforms. It can be downloaded at `http://people.inf.ethz.ch/cremersc/scyther/`, and comes with a library of example protocols modeled after the SPORE library [7].

3 Performance and Applications

We have extensively investigated the performance of Scyther compared to other state-of-the-art protocol verification tools, which is reported in [9]. In these tests, Scyther outperformed the state-of-the-art Avispa tools [2]. Even though no abstraction techniques are used by Scyther, it offered a level of performance similar to the abstraction-based ProVerif tool [1]. In practice this means small (e.g. Needham-Schroeder, Yahalom, Otway-Rees) to medium-sized (e.g. TLS, Kerberos) protocols are usually verified in less than a second. To the best of our knowledge, Scyther is currently the fastest protocol verification tool that does not use approximation methods.

Scyther has been successfully used for the analysis and design of protocols, and has also been used for theoretical research and teaching. Exploiting the state-of-the-art performance of Scyther, we have discovered a number of previously unreported attacks, e.g. as in [10, 3]. Scyther has also been used to verify theoretical results regarding protocol composition in [11], and was used for finding the counterexample that led to the main theorem of [12]. The tool

is currently being used for teaching purposes at several universities, including the Eindhoven University of Technology, ETH Zurich, University of Luxembourg, University of Twente, and the University of Grenoble. For teaching, the clear relation between the protocol specification and the protocol semantics has proven useful in explaining the fundamentals of protocol design and analysis. The concise protocol descriptions help in focussing on the protocol as opposed to tool details, in contrast to other protocol tools, which require the specification of error-prone scenarios for the verification of properties. For teaching purposes, a set of example exercises for students is available at `http://people.inf.ethz.ch/cremersc/scyther/scyther-exercises.html`.

In future work we aim to turn the informal Scyther proof output into a proof object that can be verified by mechanical theorem provers. The underlying protocol model has already been modeled in Isabelle/HOL, as described in [13].

References

1. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: Proc. 14th IEEE Computer Security Foundations Workshop (CSFW), pp. 82–96. IEEE, Los Alamitos (2001)
2. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, L., Drielsma, P., Heám, P., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
3. Cremers, C.: Feasibility of multi-protocol attacks. In: Proc. of The 1st Int. Conf. on Availability, Reliability and Security (ARES), pp. 287–294. IEEE, Los Alamitos (2006)
4. Cremers, C., Mauw, S.: Operational semantics of security protocols. In: Leue, S., Systä, T.J. (eds.) Scenarios: Models, Transformations and Tools. LNCS, vol. 3466, pp. 66–89. Springer, Heidelberg (2005)
5. Song, D.: An Automatic Approach for Building Secure Systems. PhD thesis, UC Berkeley (December 2003)
6. Doghmi, S., Guttman, J.D., Thayer, F.: Skeletons, homomorphisms, and shapes: Characterizing protocol executions. In: Proc. of the 23rd Conf. on the Mathematical Foundations of Programming Semantics (MFPS XXIII). ENTCS, vol. 173, pp. 85–102. Elsevier ScienceDirect, Amsterdam (2007)
7. Security Protocols Open Repository, `http://www.lsv.ens-cachan.fr/spore`
8. Cremers, C.: Scyther - Semantics and Verification of Security Protocols. Ph.D. dissertation, Eindhoven University of Technology (2006)
9. Cremers, C., Lafourcade, P.: Comparing state spaces in automatic protocol verification. In: Proc. of the 7th Int. Workshop on Automated Verification of Critical Systems (AVoCS 2007). ENTCS (September 2007) (to appear)
10. Cremers, C., Mauw, S.: Generalizing Needham-Schroeder-Lowe for multi-party authentication, CSR 06-04, Eindhoven University of Technology (2006)

11. Andova, S., Cremers, C., Gjøsteen, K., Mauw, S., Mjølsnes, S., Radomirović, S.: A framework for compositional verification of security protocols. Information and Computation 206, 425–459 (2008)
12. Cremers, C.: On the protocol composition logic PCL. In: Abe, M., Gligor, V. (eds.) Proc. of the ACM Symposium on Information, Computer & Communication Security (ASIACCS 2008), Tokyo, pp. 66–76. ACM Press, New York (2008)
13. Meier, S.: A formalization of an operational semantics of security protocols. Diploma thesis, ETH Zurich (August 2007), `http://people.inf.ethz.ch/meiersi/fossp/index.html`