

Network Security: Protocol verification

Ben Smyth

March 27, 2007

“a mathematical proof is the search for knowledge that is more absolute than the knowledge accumulated by any other discipline”
- Simon Singh, Fermat's last theorem.

Proposition: Proving security properties is essential for the development of secure protocols

Motivation: Numerous protocol failures

Today's objectives:

- Introduce current proof techniques
- Argue why traditional methodologies are insufficient
- Put forward formal methods as a complementary tool
- Demonstrate the practicality of one such tool, namely Proverif

Unconditional security (information theoretic security): The attacker has unbound computational power

Computational security: The attacker has bound computational power

- **Complexity-theoretic security:** (*...ask me next week...*)
- **Provable security:** Difficulty of defeating the system is shown to be as difficult as *“solving a well-known and supposedly difficult problem.”* [HAC]
- **Practical security:** the best known attack is known to be beyond the computational resources of the adversary

Provable security is insufficient

- Effective attacks do not solve difficult problems (for example integer factorisation), they discover weaknesses in the protocol
- Formalistic proofs are unreadable, (so guess what) no one reads them. As a consequence proof checking is unmet objective.

Definition of formal methods (adapted from Catherine Meadows):

Formal methods provide a mechanism to model the protocol and its requirements, together with a procedure for proving correctness. Essentially ignoring analytical, reduction-based proofs found in the provable security class.

Dolev & Yao model (1970s)

Meadow's (1992) classifies four classes of formal verification tools:

- Standard languages and verification tools (e.g. Isabelle, CSP)
- Expert systems (e.g. Interegator)
- Modal logic (e.g. BAN logic)
- State spaces (e.g. NRL)

Subsequently Meadow's (1995) added

- Algebraic systems (e.g. Applied pi calculus)

More recent additions (late 1990s/early 2000):

- Hybrid state machines/algebraic systems (e.g. Proverif)
- Strand spaces
- Type checkers

- Perfect cryptography
- Attacker controls the network (delivers messages)
 - Eavesdrop
 - Replay
 - Modify
 - Inject

Concurrent system: two or more processes that run simultaneously and involve some form of interaction.

Process calculus: high level language for modelling a concurrent system.

History of the applied pi calculus:

- **1970s:** Milner's Calculus of Communicating Systems (CCS)
- **1989:** Milner *et al.* continuation of CCS defining pi calculus
- **1999:** Abadi & Gordon produced spi calculus
- **2001:** Abadi & Fournet developed the applied pi calculus

Proverif introduced by Bruno Blanchet

Live demo