Automated Security Protocol Analysis with AVISPA

Jim Owens
Advisor: Chris Lynch
Overview

• Motivation & background

• The AVISPA tool
  – Architecture
  – High-level Protocol Specification Language (HLPSL)
  – Analysis of protocol models
  – Interpretation of results
  – Some limitations

• Conclusions
Motivation & background

- Why automated protocol analysis?
- Definitions
- Examples
  - #1 Needham-Schroeder protocol (1978-1996)
  - #2 Wireless Equivalent Privacy protocol (1999-2001)
Why automated protocol analysis?

- Rapid growth of network-based services requires development of new security protocols
- On open networks, protocols must withstand worst-case attack scenarios
- Designing secure protocols is a hard problem
- Experience teaches that even supposedly strong protocols are shown to be flawed years--even decades--after widespread adoption
Definitions

- Security protocol: An *abstract or concrete protocol that performs a security-related function* (usually involving the exchange of messages), applying *cryptographic methods* for purposes such as
  - Key establishment or agreement
  - Entity authentication
  - Symmetric encryption/message authentication
  - Secured data transport
  - Non-repudiation
Definitions

• Symmetric encryption: A class of algorithms that uses a shared secret between two or more parties for secure communication. Requires secret key exchange.

• Public-key encryption: A class of algorithms that uses separate but related keys for encryption. Requires no secret key exchange.
Definitions

- **Nonce (number used once)**: A *random* value generated by one party and returned to that party (in some form) to show that a message is fresh.

- **Authentication**: *the act of establishing or confirming someone or something as authentic; that is, that claims made by or about the subject are true.*
Example #1: NSPK Protocol

- Needham-Schroeder Public Key Authentication Protocol (1978)

1. $A \rightarrow B$: $E_B^{N_A}(A)$
2. $B \rightarrow A$: $E_A^{N_A}(N_B)$
3. $A \rightarrow B$: $E_B^{N_B}$
Example #1: NSPK Protocol

• Lowe's attack on Needham-Schroeder, found using FDR checker (1996)

1. $A \rightarrow C$: $E_C(N_A, A)$
2. $C \rightarrow A$: $E_A(N_A, N_B)$
3. $A \rightarrow C$: $E_C(N_B)$

1'. $C_A \rightarrow B$: $E_B(N_A, A)$
2'. $B \rightarrow C$: $E_A(N_A, N_B)$
3'. $C_A \rightarrow B$: $E_B(N_B)$
Example #1: NSPK Protocol

• Lowe's fix for Needham-Schroeder, proved secure (for a finite model) using FDR (1996)

1. $A \rightarrow B: E_B(N_A, A)$

2. $B \rightarrow A: E_A(N_A, N_B, B)$

3. $A \rightarrow B: E_B(N_B)$
Example #2: WEP

- **Wireless Equivalent Protocol**
  - Included in IEEE 802.11 standard, ratified 9/1999
  - In a 2001 paper, Borisov, Goldberg & Wagner revealed critical flaws in the protocol
  - By 2001, WEP was already widely deployed; numerous attempts to improve the protocol failed
  - To date, WEP still ships on most wireless devices
  - Powerful tools for breaking WEP are freely available and easy to use, requiring no special skills
Example #2: WEP

- Basic problem with WEP
  - TCP/IP contains large amounts of known plaintext
  - 24-bit IV space quickly results in keystream reuse
    - Sequential IVs: Using 1500-byte packets at 5 Mbps, available space is exhausted < \( \frac{1}{2} \) day, at worst
    - Random IVs: collisions likely after 5,000 packets (Birthday paradox)
  - Use of XOR to encrypt: \( C = P \oplus RC4(v, k) \)

\[
C_1 = P_1 \oplus RC4(v, k) \quad \text{and} \quad C_2 = P_2 \oplus RC4(v, k)
\]

\[
C_1 \oplus C_2 = (P_1 \oplus RC4(v, k)) \oplus (P_2 \oplus RC4(v, k)) = P_1 \oplus P_2
\]
The AVISPA tool

• **Automated Validation of Internet Security Protocols** and **Applications** project objectives:
  
  – Develop a rich specification language for formalizing protocols, security goals & threat models
  
  – Advance the state-of-the-art in automated deduction techniques
  
  – Build a tool to support automatic validation or detection of errors
  
  – Demonstrate proof-of-concept on a large collection of practically relevant, industrial protocols
  
  – Migrate this technology into industry standardization organizations such as the IETF
AVISPA architecture

High-Level Protocol Specification Language (HLPSL)

Translator HLPSL2IF

Intermediate Format (IF)

On-the-fly Model-Checker (OFMC)
CL-based Attack Searcher (CL-AtSe)
SAT-based Model-Checker (SATMC)
Tree-Automata-based Protocol Analyzer (TA4SP)

Output
HLPSL

- Role-based formal language for specifying
  - Control-flow patterns
  - Data structures
  - Cryptographic operators and their algebraic properties (limited)
  - Alternative intruder models
  - Security properties of protocols, assuming perfect cryptography
HLPSL model format

• HLPSL is role-based:
  – Agent roles, including starting parameters, local variables, and transitions
  – Session role, including starting parameters, local variables, and composition of agents
  – Environment role, including constants, intruder knowledge, composition of sessions, and security goals
HLPSL agent role

role alice ( A, B : agent, Ka, Kb : public_key, SND, RCV : channel (dy))

played_by A def=

  local State : nat,
  Na, Nb: text

init State := 0

transition

  0. State = 0 \ RCV(start) =>
     State' := 2 \ Na' := new() \ SND({Na'.A}_Kb)
       \ secret(Na',na,{A,B})
       \ witness(A,B,bob_alice_na,Na')

  2. State = 2 \ RCV({Na.Nb'}_Ka) =>
     State' := 4 \ SND({Nb'}_Kb)
       \ request(A,B,alice_bob_nb,Nb')

end role
role session( A, B : agent,
        Ka, Kb: public_key)
def=
    local SA, RA, SB, RB: channel (dy)
    composition
        alice(A,B,Ka,Kb,SA,RA)
        \ bob (A,B,Ka,Kb,SB,RB)
end role
HLPSL environment role

role environment()
def=
  const a, b : agent,
  ka, kb, ki : public_key,
  na, nb,
  alice_bob_nb,
  bob_alice_na : protocol_id
intruder_knowledge = {a, b, ka, kb, ki, inv(ki)}
composition
  session(a,b,ka,kb)
  \ session(a,i,ka,ki)
  \ session(i,b,ki,kb)
end role
HLPSL goals

goal

  secrecy_of na, nb
  authentication_on alice_bob_nb
  authentication_on bob_alice_na

end goal
Analysis of protocol models

- HLPSL models translate to low-level spec, *Intermediate Format* (IF), suitable for automatic analysis using rewrite rules
- AVISPA provides four back-ends for analysis
  - Constraint-Logic-based Attack Searcher (CL-AtSe)
  - On-the-fly Model-Checker (OFMC)
  - SAT-based Model-Checker (SATMC)
  - TA4SP (Tree Automata based on Automatic Approximations for Analysis of Security Protocols)
Interpretation of results

ATTACK TRACE

i -> (a,6): start
(a,6) -> i: \{Na(1).a\}_ki
i -> (b,3): \{Na(1).a\}_kb
(b,3) -> i: \{Na(1).Nb(2)\}_ka
i -> (a,6): \{Na(1).Nb(2)\}_ka
(a,6) -> i: \{Nb(2)\}_ki
i -> (i,17): Nb(2)
i -> (i,17): Nb(2)

1. \(A \rightarrow C: E_C(N_A, A)\)
1'. \(C_A \rightarrow B: E_B(N_A, A)\)
2'. \(B \rightarrow C: E_A(N_A, N_B)\)
2. \(C \rightarrow A: E_A(N_A, N_B)\)
3. \(A \rightarrow C: E_C(N_B)\)
3'. \(C_A \rightarrow B: E_B(N_B)\)
Applying Lowe's fix

1. $A \rightarrow B: E_B (N_A, A)$

2. $B \rightarrow A: E_A (N_A, N_B, B)$

3. $A \rightarrow B: E_B (N_B)$
Applying Lowe's fix

- Bob's role

transition

1. State = 1 \[ \land \] RCV({Na'.A}_Kb) =|> State':= 3 \[ \land \] Nb' := new() \[ \land \] SND({Na'.Nb'.B}_Ka) \[ \land \] secret(Nb',nb,{A,B}) \[ \land \] witness(B,A,alice_bob_nb,Nb')

3. State = 3 \[ \land \] RCV({Nb}_Kb) =|> State':= 5 \[ \land \] request(B,A,bob_alice_na,Na)
Applying Lowe's fix

- Alice's role

  transition

  0. State = 0 /\ RCV(start) =|>
     State' := 2 /\ Na' := new() /\ SND({Na'.A}_Kb)
               /\ secret(Na',na,{A,B})
               /\ witness(A,B,bob_alice_na,Na')

  2. State = 2 /\ RCV({Na.Nb'.B}_Ka) =|>
     State' := 4 /\ SND({Nb'}_Kb)
               /\ request(A,B,alice_bob_nb,Nb')
Limitations

● In general, some aspects of protocols can not be modeled effectively; for example, probability

● HLPSL parser can miss subtle syntax errors, resulting in models that produce misleading results

● Some cryptographic primitives are not modeled effectively in AVISPA; for example, timestamps and nonce lifetimes
Conclusions

- Jim's guidelines for developing new protocols
  
  #1 Don't do it!
  
  #2 See guideline #1
  
  #3 Don't try to start from scratch. Use a tested existing protocol and modify it
  
  #4 Follow Abadi & Needham's *Prudent Engineering Practice for Cryptographic Protocols*
  
  #5 Use AVISPA or another tool to find errors
  
  #6 Get a thorough review by security community
  
  #7 Still here? See guideline #1