MSR

Iliano Cervesato  iliako@itt.nrl.navy.mil

ITT Industries, Inc  @  NRL - Washington DC

http://www.cs.stanford.edu/~iliano/

Dept. of Information Science  東京大学 University of Tokyo  March 12th, 2001
Outline

I. Mis-specification languages

II. MSR
   - Overview
   - Typing
   - Access control
   - Execution
   - Properties
   - Example

III. The most powerful attacker
   - Dolev-Yao intruder
Part I

Mis-Specification Languages
Why is Protocol Analysis Difficult?

- Subtle cryptographic primitives
  - Dolev-Yao abstraction

- Distributed hostile environment
  - “Prudent engineering practice”

- Inadequate specification languages
  - ... the devil is in details ...
Dolev-Yao Abstraction

- Symbolic data
  - No bit-strings

- Perfect cryptography
  - No guessing of keys

- Public knowledge soup
  - Magic access to data
Languages to Specify What?

- Message flow
- Message constituents
- Operating environment
- Protocol goals
Desirable Properties

- Unambiguous
- Simple
- Flexible
  - Adapts to protocol
- Powerful
  - Applies to a wide class of protocols
- Insightful
  - Gives insight about protocols
“Usual Notation”

A → B: \{n_A, A\}_{kB}

B → A: \{n_A, n_B\}_{kA}

A → B: \{n_B\}_{kB}
How does it do?

- **Flow**
  - Expected run
- **Constituents**
  - Side remarks
- **Environment**
  - Side remarks
- **Goals**
  - Side remarks

<table>
<thead>
<tr>
<th>Unambiguous</th>
<th>Simple</th>
<th>Flexible</th>
<th>Powerful</th>
<th>Insightful</th>
</tr>
</thead>
<tbody>
<tr>
<td>🙁</td>
<td>😊</td>
<td>😊</td>
<td>😊</td>
<td>😞</td>
</tr>
</tbody>
</table>
Strands

\[
\{n_A, A\}_{kB} \rightarrow \rightarrow \{n_A, A\}_{kB}
\]

\[
\{n_A, n_B\}_{kA} \leftarrow \leftarrow \{n_A, n_B\}_{kA}
\]

\[
\{n_B\}_{kB} \rightarrow \rightarrow \{n_B\}_{kB}
\]
How do they do?

- Flow
  - Role-based
- Constituents
  - Informal math.
- Environment
  - Side remarks
- Goals
  - Side remarks

- Unambiguous 😞
- Simple 😊
- Flexible 😞
- Powerful 😞
- Insightful 😊
MSR 1.x - Initiator

\[ \pi_{A0}(A) \rightarrow L_0(A), \pi_{A0}(A) \]

\[ L_0(A), \pi_{A1}(B) \rightarrow \exists n_A. L_1(A,B,n_A), N({n_A,A}_{kB}), \pi_{A1}(B) \]

\[ L_1(A,B,n_A), N({n_A,n_B}_{KA}) \rightarrow L_2(A,B,n_A,n_B) \]

\[ L_2(A,B,n_A,n_B) \rightarrow L_3(A,B,n_A,n_B), N({n_B}_{kB}) \]

where

\[ \pi_{A0}(A) = Pr(A), PrvK(A,k_A^{-1}) \]

\[ \pi_{A1}(B) = Pr(B), PubK(B,k_B) \]
MSR 1.x - Responder

\[ \pi_{B0}(B) \rightarrow L_0(B), \pi_{B0}(B) \]

\[ L_0(A), \pi_{B1}(A), N({n_A, A}_{k_B}) \rightarrow L_1(A, B, n_A), \pi_{B1}(A) \]

\[ L_1(A, B, n_A) \rightarrow \exists n_B. L_2(A, B, n_A, n_B), N({n_A, n_B}_{k_A}) \]

\[ L_2(A, B, n_A, n_B), N({n_B}_{k_B}) \rightarrow L_3(A, B, n_A, n_B) \]

where

\[ \pi_{B0}(B) = Pr(B), \ PrvK(B, k_B^{-1}) \]

\[ \pi_{B1}(A) = Pr(A), \ PubK(A, k_A) \]

MSR, a Framework for Security Protocols and their Meta-Theory
How did we do?

- Flow
  - Role-based
- Constituents
  - Persistent info.
- Environment
  - In part
- Goals

- Unambiguous
- Simple
- Flexible
- Powerful
- Insightful
How *will* we do?

- **Flow**
  - Role-based
- **Constituents**
  - Strong typing
- **Environment**
  - *In part*

- **Goals**

- **Unambiguous**
- **Simple**
- **Flexible**
- **Powerful**
- **Insightful**
Part II
MSR
What’s in MSR 2.0?

• Multiset rewriting with existentials
• Dependent types w/ subsorting
• Memory predicates
• Constraints
Terms

- Atomic terms
  - Principal names $A$
  - Keys $k$
  - Nonces $n$
  - ...

- Term constructors
  - $(_ _)$
  - $\{\} _ \{\{\}\} _$
  - $[\_] _$
  - ...

MSR, a Framework for Security Protocols and their Meta-Theory
Rules

∀x₁: τ₁.
  ...
∀xₙ: τₙ.

lhs → rhs

∃y₁: τ₁'.
  ...
∃yₙ: τₙ'.

- N(t) Network
- L(t, ..., t) Local state
- Mₐ(t, ..., t) Memory
- χ Constraints

Network

Local state

Memory

Constraints

MSR, a Framework for Security Protocols and their Meta-Theory
Types of Terms

- A: princ
- n: nonce
- k: shK A B
- k: pubK A
- k': privK k
- ... (definable)

Types can depend on term

- Captures relations between objects
- Subsumes persistent information
  - Static
  - Local
  - Mandatory
Subtyping

\[ \tau :: \text{msg} \]

- Allows atomic terms in messages
- Definable
  - Non-transmittable terms
  - Sub-hierarchies
Role state predicates

$L_1(A, \tau, \ldots, \tau)$

- Hold data local to a role instance
  - Lifespan = role

- Invoke next rule
  - $L_1 = \text{control}$
  - $(A, \tau, \ldots, \tau) = \text{data}$
Memory Predicates

\[ M_A(t, \ldots, t) \]

- Hold private info. across role exec.
- Support for subprotocols
  - Communicate data
  - Pass control
- Interface to outside system
- Implements intruder
Constraints

- Guards over interpreted domain
  - Abstract
  - Modular
- Invoke constraint handler
- E.g.: timestamps
  - \((T_E = T_N + T_d)\)
  - \((T_N < T_E)\)
Type of predicates

- Dependent sums

\[ \sum_{x: \tau} \tau \]

- Forces associations among arguments

E.g.: \( \text{princ}(A) \times \text{pubK} A(k_A) \times \text{privK} k_A \)
Roles

- Generic roles

\[ \exists L: \tau'_1(x_1) \times \cdots \times \tau'_n(x_n) \]

\[ \forall x: \tau. \quad \text{lhs} \quad \exists y: \tau'. \quad \rightarrow \quad \text{rhs} \]

- Anchored roles

\[ \exists L: \tau'_1(x_1) \times \cdots \times \tau'_n(x_n) \]

\[ \forall x: \tau. \quad \text{lhs} \quad \exists y: \tau'. \quad \rightarrow \quad \text{rhs} \]

Role state pred. var. declarations

Role owner

MSR, a Framework for Security Protocols and their Meta-Theory
∀A

∃L: princ × princ(B) × pubK B × nonce.

∀B: princ
∀kB: pubK B
• → ∃nA: nonce.
L(A,B,kB,nA) N({nA,A}kB)

∀...
∀kA: pubK A
∀k'A: privK kA
∀nA,nB: nonce
L(A,B,kB,nA) N({nA,nB}kA) → N({nB}kB)
MSR 2.0 – NS Responder

\[ \exists \mathcal{L} : \text{princ}^{(B)} \times \text{princ}^{(A)} \times \text{pubK}^{B(k_B)} \times \text{privK}^{k_B} \]
\[ \times \text{nonce} \times \text{pubK}^{A} \times \text{nonce}. \]

\[ \forall k_B : \text{pubK}^{B} \]
\[ \forall k'_B : \text{privK}^{k_B} \]
\[ \forall A : \text{princ} \]
\[ \forall n_A : \text{nonce} \]
\[ \forall k_A : \text{pubK}^{A} \]
\[ \forall \ldots \]
\[ \forall n_B : \text{nonce} \]
\[ \mathcal{L}(B,k_B,k'_B,A,n_A,k_A,n_B) \]
\[ \rightarrow \mathcal{L}(\ldots) \]
\[ \text{N}({n_B}_{k_B}) \rightarrow \text{N}({n_A,n_B}_{k_A}) \]

MSR, a Framework for Security Protocols and their Meta-Theory
Type Checking

\[ \Sigma \vdash P \]

\[ \Gamma \vdash t : \tau \]

- **Catches:**
  - Encryption with a nonce
  - Transmission of a long term key
  - Circular key hierarchies, ...

- **Static and dynamic uses**

- **Decidable**
Access Control

- Catches
  - A signing/encrypting with B’s key
  - A accessing B’s private data, ...

- Fully static
- Decidable
- Gives meaning to Dolev-Yao intruder
Snapshots

$$C = [S]^R \sum$$

**State**
- $N(t)$
- $L_1(t, \ldots, t)$
- $M_A(t, \ldots, t)$

**Signature**
- $a : \tau$
- $L_1 : \tau$
- $M_\pi : \tau$

Active role set
Execution Model

- Activate roles
- Generates new role state pred. names
- Instantiate variables
- Apply rules
- Skips rules

\[ P \triangleright C \rightarrow C' \]

1-step firing

MSR, a Framework for Security Protocols and their Meta-Theory
Rule application

\[ F, \chi \rightarrow \exists n : \tau. \ G(n) \]

- Constraint check
  \[ \Sigma \models \chi \quad \text{(constraint handler)} \]

- Firing
  \[
  \left[ S_1 \right]^R_{\Sigma} \quad \rightarrow \quad \left[ S_2 \right]^R_{\Sigma, c : \tau} \quad \text{c not in } S_1
  \]

\[ S, F \quad \rightarrow \quad S, G(c) \]

MSR, a Framework for Security Protocols and their Meta-Theory
Properties

- Admissibility of parallel firing
- Type preservation
- Access control preservation
- Completeness of Dolev-Yao intruder

New
Completed Case-Studies

- Full Needham-Schroeder public-key
- Otway-Rees
- Neuman-Stubblebine repeated auth.
- OFT group key management
- Dolev-Yao intruder
Part III

The Most Powerful Attacker
Execution with an Attacker

\[ P, P_I \triangleright C \rightarrow C' \]

- Selected principal(s): \( I \)
- Generic capabilities: \( P_I \)
  - Well-typed
  - AC-valid

- Modeled completely within MSR

MSR, a Framework for Security Protocols and their Meta-Theory
The Dolev-Yao Intruder

- Specific protocol suite $P_{DY}$
- Underlies every protocol analysis tool
- Completeness still unproved !!!
Capabilities of the D-Y Intruder

- Intercept / emit messages
- Split / form pairs
- Decrypt / encrypt with known key
- Look up public information
- Generate fresh data
DY Intruder – Data access

- $M_I(t)$ : Intruder knowledge

\[
\left( \forall A: \text{princ.} \implies M_I(A) \right)^I
\]

\[
\left( \forall A: \text{princ} \implies M_I(k) \right)^I \text{ + dual}
\]

\[
\left( \forall A: \text{princ} \implies M_I(k) \right)^I \left( \forall k: \text{pubK} A \implies M_I(k') \right)^I
\]

- No nonces, no other keys, ...

MSR, a Framework for Security Protocols and their Meta-Theory
DY Intruder – Data Generation

• Safe data
\[
\left( \cdot \rightarrow \exists n:\text{nonce. } M_\mathcal{I}(n) \right)^\mathcal{I} \quad \left( \cdot \rightarrow \exists m:\text{msg. } M_\mathcal{I}(m) \right)^\mathcal{I}
\]

• Anything else?
\[
\left( \forall A,B:\text{princ. } \cdot \rightarrow \exists k:\text{shK } A B. M_\mathcal{I}(k) \right)^\mathcal{I} \quad ???
\]

• It depends on the protocol !!!
  ➢ Automated generation ?
DY Intruder Stretches AC to Limit

\[
\text{Well-typed} \rightarrow \text{AC-valid} \rightarrow \text{Dolev-Yao intruder}
\]
Completeness of D-Y Intruder

• If \( P \triangleright [S]^R_\Sigma \rightarrow [S']^{R'}_{\Sigma'} \)
  with all well-typed and AC-valid

• Then

\[ P, P_{DY} \triangleright [S]^R_\Sigma \rightarrow [S']^{R'}_{\Sigma'} \]
Encoding of $P$, $S$, $\Sigma$

$P$  Remove roles anchored on $I$

$S$  Map $I$’s state / mem. pred. using $M_I$

$\Sigma$  Remove $I$’s role state pred.; add $M_I$
Encoding of $R$

- No encoding on structure of $R$
  - Lacks context!

- Encoding on AC-derivation for $R$
  
  \[ A ::= \sum \parallel \leftarrow R \]

  - Associate roles from $P_{DY}$ to each AC rule
Completeness proof

- Induction on execution sequence
- Simulate every step with $P_{DY}$
  - Rule application
    - Induction on AC-derivation for $R$
    - Every AC-derivation maps to execution sequence relative to $P_{DY}$
  - Rule instantiation
    - AC-derivations preserved
    - Encoding unchanged
Consequences

- Justifies design of current tools
- Support optimizations
  - D-Y intr. often too general/inefficient
    - Generic optimizations
    - Per protocol optimizations
    - Restrictive environments
- Caps multi-intruder situations
Conclusions

• Framework for specifying protocols
  - Precise
  - Flexible
  - Powerful

• Provides
  - Type /AC checking
  - Sequential / parallel execution model
  - Insights about Dolev-Yao intruder
Future work

• Experimentation
  - Clark-Jacob library
  - Fair-exchange protocols
  - More multicast

• Pragmatics
  - Type-reconstruction
  - Operational execution model(s)
  - Implementation

• Automated specification techniques