MSR

Iliano Cervesato iliano@itd.nrl.navy.mil
ITT Industries, Inc @ NRL - Washington DC

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

Dept. of Computer Science - UMBC April 23rd, 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”

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

\[ B \rightarrow A: \{n_A, n_B\}_{kA} \]

\[ A \rightarrow B: \{n_B\}_{kB} \]
### How does it do?

<table>
<thead>
<tr>
<th>Flow</th>
<th>• Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Expected run</td>
</tr>
<tr>
<td></td>
<td>• Constituents</td>
</tr>
<tr>
<td></td>
<td>• Environment</td>
</tr>
<tr>
<td></td>
<td>• Goals</td>
</tr>
<tr>
<td></td>
<td>• Unambiguous</td>
</tr>
<tr>
<td></td>
<td>• Simple</td>
</tr>
<tr>
<td></td>
<td>• Flexible</td>
</tr>
<tr>
<td></td>
<td>• Powerful</td>
</tr>
<tr>
<td></td>
<td>• Insightful</td>
</tr>
</tbody>
</table>

- Flow: Expected run
- Constituents: Side remarks
- Environment: Side remarks
- Goals: Side remarks
- Unambiguous
- Simple
- Flexible
- Powerful
- Insightful
Strands

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

\[ \{n_A, n_B\}_{kA} \rightarrow \{n_B\}_{kB} \]

\[ \{n_A, A\}_{kB} \rightarrow \{n_A, n_B\}_{kA} \rightarrow \{n_B\}_{kB} \]
How do they do?

- **Flow**
  - Role-based

- **Constituents**
  - Informal math.

- **Environment**
  - Side remarks

- **Goals**
  - Side remarks

- **Unambiguous**
  - Simple
  - Powerful
  - Insightful

**MSR, a Framework for Security Protocols and their Meta-Theory**
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, a Framework for Security Protocols and their Meta-Theory
MSR 1.x - Responder

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

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

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

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

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 😊

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

MSR
What’s in MSR 2.0?

- Multiset rewriting with existentials
- Dependent types with 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.

∀x₁: τ₁.

... 

∀xₙ: τₙ.

lhs \rightarrow rhs

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

• N(t) Network
• L(t, ..., t) Local state
• Mₐ(t, ..., t) Memory
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

MSR, a Framework for Security Protocols and their Meta-Theory
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
  \[ \tau(x) \times \tau \]
  
  - Forces associations among arguments

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

- **Generic roles**

\[
\exists L: \tau'_1(x_1) \times \cdots \times \tau'_n(x_n) \ \
\forall x: \tau. \ \text{lhs} \rightarrow \exists y: \tau'. \ \text{rhs}
\]

- **Anchored roles**

\[
\exists L: \tau'_1(x_1) \times \cdots \times \tau'_n(x_n) \ \
\forall x: \tau. \ \text{lhs} \rightarrow \exists y: \tau'. \ \text{rhs}
\]

Role state pred. var. declarations

Role owner

MSR, a Framework for Security Protocols and their Meta-Theory
MSR 2.0 – NS Initiator

\[ \forall A \exists L : \text{princ} \times \text{princ}(B) \times \text{pubK} B \times \text{nonce.} \]

\[ \forall B : \text{princ} \forall k'_B : \text{privK} k_A \forall n_A, n_B : \text{nonce} \]

\[ \forall k_B : \text{pubK} B \]

\[ \exists n_A : \text{nonce.} \]

\[ \forall \ldots \]

\[ \forall k_A : \text{pubK} A \]

\[ \exists L(A,B,k_B,n_A) \]

\[ N(\{n_A,A\}_{k_B}) \]

\[ \exists L(A,B,k_B,n_A) \]

\[ N(\{n_A,n_B\}_{k_A}) \rightarrow N(\{n_B\}_{k_B}) \]
MSR 2.0 - NS Responder

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

\[ \forall k_B: \text{pubK } B \]
\[ \forall k'_B: \text{privK } k_B \]
\[ \forall A: \text{princ} \]
\[ \forall n_A: \text{nonce} \]
\[ \forall k_A: \text{pubK } A \]

\[ \forall \ldots \]
\[ \forall n_B: \text{nonce} \]

\[ \mathcal{L}(B,k_B,k'_B,A,n_A,k_A,n_B) \]
\[ \Rightarrow \mathcal{L}(\ldots) \]

\[ \Rightarrow \exists n_B: \text{nonce}. \]

\[ \mathcal{L}(\ldots) \]
\[ \Rightarrow \exists n_B: \text{nonce}. \]

\[ \mathcal{N}(\{n_A,A\}_{k_B}) \]
\[ \Rightarrow \exists n_B: \text{nonce}. \]

\[ \mathcal{L}(\ldots) \]
\[ \Rightarrow \exists n_B: \text{nonce}. \]

\[ \mathcal{N}(\{n_A,n_B\}_{k_A}) \]

\[ \Rightarrow \exists n_B: \text{nonce}. \]

\[ \mathcal{N}(\{n_B\}_{k_B}) \]

\[ \Rightarrow \exists n_B: \text{nonce}. \]
Type Checking

$\Sigma \vdash P$

$t$ has type $\tau$ in $\Gamma$

$\Gamma \vdash t : \tau$

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

- Static and dynamic uses

- Decidable

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

\[ C = [S]^R \Sigma \]

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

Signature
- \( a : \tau \)
- \( L_1 : \tau \)
- \( M_\_ : \tau \)

Active role set
Execution Model

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

\[ P \xrightarrow{*} C \rightarrow C' \]
Rule application

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

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

- **Firing**
  \[
  \frac{[S_1]^R}{\Sigma} \rightarrow \frac{[S_2]^R}{\Sigma, c : \tau} \quad c \ not \ in \ S_1
  \]

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

\[ P, P_I \xrightarrow{C} C' \]

- 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

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

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

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

$$\left[ \forall k: \text{pubK} \right. \left. \forall k': \text{privK} \implies M_I(k') \right]^I$$

- No nonces, no other keys, ...
DY Intruder – Data Generation

• Safe data

\[
\begin{align*}
\cdot & \rightarrow \exists n: \text{nonce}. M_I(n) \\
\cdot & \rightarrow \exists m: \text{msg}. M_I(m)
\end{align*}
\]

• Anything else?

\[
\begin{align*}
\forall A, B: \text{princ}. \cdot & \rightarrow \exists k: \text{shK} A B. M_I(k)
\end{align*}
\]

• It depends on the protocol !!!

➤ Automated generation ?

MSR, a Framework for Security Protocols and their Meta-Theory
DY Intruder Stretches AC to Limit

AC-valid

Well-typed

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 $R$

$$A ::= \Sigma \parallel \vdash R$$

- 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**