MSR

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

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

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

- **Unambiguous**
- **Simple**
- **Flexible**
- **Powerful**
- **Insightful**
Strands

\(\{n_A, A\}_{kB}\) \quad \rightarrow \quad \rightarrow \quad \{n_A, A\}_{kB}\)

\(\{n_A, n_B\}_{kA}\) \quad \leftarrow \quad \leftarrow \quad \{n_A, n_B\}_{kA}\)

\(\{n_B\}_{kB}\) \quad \rightarrow \quad \rightarrow \quad \{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, a Framework for Security Protocols and their Meta-Theory
MSR 1.x - Responder

\[ \pi_{B_0}(B) \rightarrow L_0(B), \pi_{B_0}(B) \]

\[ L_0(A), \pi_{B_1}(A), N(\{n_A, A\}_{k_B}) \rightarrow L_1(A, B, n_A), \pi_{B_1}(A) \]

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

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

where

\[ \pi_{B_0}(B) = Pr(B), PrvK(B, k_B^{-1}) \]

\[ \pi_{B_1}(A) = Pr(A), PubK(A, k_A) \]
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 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

\[ \forall x_1: \tau_1. \]
\[ \ldots \]
\[ \forall x_n: \tau_n. \]

\[ \iff \]
\[ \exists y_1: \tau'_1. \]
\[ \ldots \]
\[ \exists y_n: \tau'_n. \]

lhsp -> rhs

- N(t) Network
- L(t, ..., t) Local state
- MA(t, ..., t) Memory
- \( \chi \) 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

\[ L_\| (A, t, \ldots, t) \]

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

- Invoke next rule
  - \[ L_\| = \text{control} \]
  - \[ (A, t, \ldots, t) = \text{data} \]
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

$\chi$

- 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.: princ(A) \times pubK A(k_A) \times privK k_A
Roles

- **Generic roles**

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

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

- **Anchored roles**

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

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

Role state pred., var. declarations

Role owner

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

\[
\forall A: \text{princ} \times \text{princ}(B) \times \text{pubK} B \times \text{nonce}. \\
\exists L: \text{princ} \times \text{princ}(B) \times \text{pubK} B \times \text{nonce}. \\
\forall B: \text{princ} \Rightarrow \exists n_A: \text{nonce}. \\
\forall k_B: \text{pubK} B \Rightarrow L(A,B,k_B,n_A) \Rightarrow N(\{n_A,A\}_{k_B}) \\
\forall \ldots \\
\forall k_A: \text{pubK} A \Rightarrow L(A,B,k_B,n_A) \Rightarrow N(\{n_A,n_B\}_{k_A}) \Rightarrow N(\{n_B\}_{k_B}) \\
\forall n_A,n_B: \text{nonce}
\]
∀k_B: pubK B
∀k'_B: privK k_B
∀A: princ
∀n_A: nonce
∀k_A: pubK A
∀... 
∀n_B: nonce
N({n_B}k_B)
∀B

∃L: princ(B) x pubK B(k_B) x privK k_B x nonce.

N({n_A,A}_{k_B}) \rightarrow \exists n_B: nonce.
L(B,k_B,k'_B,n_B) N({n_A,n_B}_{k_A})

∀ ... 
∀n_B: nonce
N({n_B}_{k_B})

\rightarrow \bullet
Type Checking

\[ \Sigma \vdash P \]

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

\( t \) has type \( \tau \) in \( \Gamma \)

\( P \) is well-typed in \( \Sigma \)

- 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

\[ C = \left[ S \right]^{R}_{\Sigma} \]

**State**
- \( N(t) \)
- \( L_i(t, \ldots, t) \)
- \( M_A(t, \ldots, t) \)

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

Active role set
Execution Model

\[ P \succ C \rightarrow C' \]

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

1-step firing
Rule application

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

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

- Firing

\[
\begin{align*}
\left[ S_1 \right]^{R \Sigma} & \quad \rightarrow \quad \left[ S_2 \right]^{R \Sigma, c:\tau} \\
S, F & \quad \rightarrow \quad S, G(c) \\
& \quad \text{c not in } S_1
\end{align*}
\]
Properties

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

- Specific protocol suite $P_{DY}$
- Underlies every protocol analysis tool
- Completeness still unproved !!!

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

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

\[
\left( \forall A: \text{princ} \quad \forall k: \text{shK} \ I \ A \quad \rightarrow \quad M_I(k) \right)^I
\]

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

\[
\left( \forall k: \text{pubK} I \quad \forall k': \text{privK} k \quad \rightarrow \quad M_I(k') \right)^I
\]

• No nonces, no other keys, ...

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

• Safe data

\[
\left( \bullet \rightarrow \exists n: \text{nonce}. \ M_I(n) \right)^I \quad \left( \bullet \rightarrow \exists m: \text{msg}. \ M_I(m) \right)^I
\]

• Anything else?

\[
\left( \forall A,B: \text{princ.} \; \bullet \rightarrow \exists k: \text{shK} \; A \; B. \; M_I(k) \right)^I \quad ???
\]

• It depends on the protocol !!!

- Automated generation?
DY Intruder Stretches AC to Limit

AC-valid

Well-typed

Dolev-Yao intruder

MSR, a Framework for Security Protocols and their Meta-Theory
Completeness of D-Y Intruder

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

• Then

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

\( P \) Remove roles anchored on \( I \)

\( S \) Map \( I \)’s state / mem. pred. using \( M_I \)

\( Σ \) Remove \( I \)’s role state pred.; add \( M_I \)
Encoding of $\mathcal{R}$

- No encoding on structure of $\mathcal{R}$
  - Lacks context!

- Encoding on AC-derivation for $\mathcal{R}$
  \[
  A :: \Sigma \parallel \vdash \mathcal{R}
  \]
  - Associate roles from $\mathcal{P}_{DY}$ 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