The Wolf Within

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

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

WITS’02
January 14th, 2002
Outline

I. Protocol specification

II. MSR in brief
   - Data Access Specification
   - Dolev-Yao intruder

III. DAS $\rightarrow$ DY Intruder

IV. Protocol Spec. $\rightarrow$ DAS

The Wolf Within
Data Access vs. Attacker

Data access policy

- Lax
- Strict

Explicit Implicit

Spi

MSR

Most languages

The Wolf Within
A 1D View

Strict language
Implicit attacker

Lax language
Explicit attacker

Language lacks power

Language has redundancy

Intro.
MSR
DAS → DY
Spec. → DAS

The Wolf Within
The Extremes

Lax and Explicit

Expressible
Reasonable
Attackable

Strict and Implicit

Expressible
= Reasonable
= Attackable

Provably empty in MSR
Summary

Strictness
+ Self-contained
+ Express what we want
− More complex

Explicitness
+ Accommodate weak attackers
− External

The option of explicitness is valuable
MSR

- Follows the Dolev-Yao abstraction
- Based on
  - Multiset rewriting, linear logic
  - Type theory
- Used to prove
  - Undecidability of protocol verification
  - Completeness of Dolev-Yao intruder
- Specifications
  - So many protocols ... so little time ...
- Related to CIL, strands, spi-calculus

Intro.
MSR
DAS → DY
Spec. → DAS
Concl.
What’s in MSR 2.0?

- Multiset rewriting with existentials
- Dependent types with subsorting
- Memory predicates
- Constraints

New
Roles

- **Generic roles**
  \[
  \exists \mathcal{L}: \tau'_1(x_1) \times \ldots \times \tau'_n(x_n)
  \]
  \[
  \forall x: \tau. \text{lhs} \quad \exists y: \tau'. \text{rhs}
  \]
  \[
  \forall x: \tau. \text{lhs} \rightarrow \exists y: \tau'. \text{rhs}
  \]

- **Anchored roles**
  \[
  \exists \mathcal{L}: \tau'_1(x_1) \times \ldots \times \tau'_n(x_n)
  \]
  \[
  \forall x: \tau. \text{lhs} \rightarrow \exists y: \tau'. \text{rhs}
  \]

**Role owner**

**Role state pred. var. declarations**

Intro.
MSR
DAS → DY
Spec. → DAS
Concl.
Rules

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

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

_lhs \quad \rightarrow \quad rhs

- **N(t)**  Network
- **L(t, ..., t)**  Local state
- **M_A(t, ..., t)**  Memory
- **\chi**  Constraints

- **N(t)**  Network
- **L(t, ..., t)**  Local state
- **M_A(t, ..., t)**  Memory
NS Initiator

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

\[ \forall B: \text{princ} \quad \forall k_B: \text{pubK}B \quad \Rightarrow \quad \exists n_A: \text{nonce}. \]

\[ \exists L(A, B, k_B, n_A): N(\{n_A, A\}_{k_B}) \]

\[ \forall k'_A: \text{privK}k_A \]

\[ \forall n_A, n_B: \text{nonce} \]

\[ \forall A \]

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

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

\[ A \rightarrow B: \{n_B\}_{k_B} \]
NS Responder

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

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

\[ \forall \ldots \]
\[ \forall n_B: \text{nonce} \]
\[ N(\{n_A, A\}_{k_B}) \rightarrow \exists n_B: \text{nonce}. \]
\[ \nabla(B, k_B, k'_B, n_B) \]
\[ N(\{n_A, n_B\}_{k_A}) \]

\[ \nabla(B, k_B, k'_B, n_B) \rightarrow \nabla(B, k_B, k'_B, n_B) \rightarrow \]
**Type Checking**

- \( \Sigma \vdash P \)
- \( \Gamma \vdash \dagger : \tau \)
- \( \dagger \) 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, ...
Data Access Specification

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

- **Static and decidable**

- **Gives meaning to**
  - the Dolev-Yao intruder

\[ \Sigma \vdash P \]

\[ \Gamma \vdash_A r \]

\( r \) is DAS-valid for \( A \) in \( \Gamma \)

\( P \) is DAS-valid in \( \Sigma \)
... pictorially
An Overview of DAS

- Interpret incoming information
  - Collect received data
  - Access unknown data

- Construct outgoing information
  - Generate data
  - Use known data
  - Access new data

... all along, verify access to data
Verifying a Rule

Knowledge set:
Collects what A knows

Context

\[ \Gamma \parallel_- A \text{ lhs } \rightarrow \Delta \]

\[ \Gamma ; \Delta \parallel_- A \text{ rhs } \]

\[ \Gamma \parallel_- A \text{ lhs } \rightarrow \text{ rhs} \]

Role owner
The Dolev-Yao Intruder Model

- Interpret incoming information
  - Collect received data
  - Access unknown data

- Construct outgoing information
  - Generate data
  - Use known data
  - Access new data

- Same operations as DAS!
Accessing Principal Names

\[ \Gamma, B: \text{princ} \vdash A \Box B \]

\[ \{ \forall B: \text{princ.} \quad \bullet \quad \rightarrow \quad M_I(B) \} \]
What did we do?

RHS data access:

- Instantiate acting principal to \( I \)
- Accessed data \( \rightarrow \) Intruder knowledge
- Meta-variables \( \rightarrow \) Rule variables
- Context provides types
Checking it out: Shared Keys

\[ \Gamma, A:\text{princ}, B:\text{princ}, k:\text{shK} \quad A \Rightarrow B \quad \parallel \quad A \Rightarrow k \]

\( \forall B: \text{princ} \)
\( \forall k: \text{shK} \quad I \quad B \quad \Rightarrow \quad M_I(k) \)

+ dual
Getting Confident: Pub./Priv. Keys

\[\Gamma, B: \text{princ}, k: \text{pubK} B \vdash A \Rightarrow k\]

\[\forall B: \text{princ} \quad \forall k: \text{pubK} B \quad \rightarrow M_I(k)\]

\[\Gamma, A: \text{princ}, k: \text{pubK} A, k': \text{privK} k \vdash A \Rightarrow k'\]

\[\forall k: \text{pubK} I \quad \forall k': \text{privK} k \quad \rightarrow M_I(k')\]
Constructing Messages: Pairs

\[ \forall t_1, t_2 : \text{msg. } M_I(t_1), M_I(t_2) \rightarrow M_I((t_1, t_2)) \]

\[ \Gamma; \Delta \vdash t_1 \quad \Gamma; \Delta \vdash t_2 \]

\[ \Gamma; \Delta \vdash (t_1, t_2) \]

\[ \Gamma |\!\!\!| - t_1 : \text{msg} \quad \Gamma |\!\!\!| - t_2 : \text{msg} \]

\[ \Gamma |\!\!\!| - (t_1, t_2) : \text{msg} \]
Now, what did we do?

RHS message construction:

- Instantiate acting principal to I
- Meta-variables $\mapsto$ Rule variables
- Premises $\mapsto$ antecedent
- Conclusion $\mapsto$ consequent
- Types from auxiliary typing derivation
Carrying on: Shared-Key Encrypt.

\[ \Gamma; \Delta \vdash \top \quad \Gamma; \Delta \vdash k \]

\[ \Gamma; \Delta \vdash \{\top\}_k \]

\[ \forall A, B: \text{princ} \]
\[ \forall k: \text{shK} A B \quad M_I(\top), M_I(k) \rightarrow M_I({\top}_k) \]
\[ \forall t: \text{msg} \]

Similar for public-key encryption
Generating Nonces

\[(\Gamma, x:nonce); (\Delta, \text{x}) \vdash \text{rhs} \]

\[
\Gamma; \Delta \vdash \exists x:nonce. \text{rhs}
\]

\[
\begin{align*}
\bullet & \rightarrow \exists x:nonce. M_{I}(x) \\
\end{align*}
\]

Similarly for other generated data
Now, what did we do?

Data generation on the RHS:

- Instantiate acting principal to $I$
- Auxiliary typing derivation gives types
- Remember generated object
- Follow knowledge acquisition flow
Accessing Shared Keys on the LHS

\[
(\Gamma, k:shK I B); \Delta \parallel \ll I k \gg (\Delta, k)
\]

Similarly for other keys

\[
\left\{ \begin{align*}
\forall B: \text{princ} \\
\forall k: \text{shK} I B
\end{align*} \right\} \rightarrow \mathcal{M}_I(k)
\]
Now, what did we do?

LHS data access:

- Instantiate acting principal to I
- Meta-variables $\rightarrow$ Rule variables
- Types from auxiliary typing derivation
- Follow knowledge acquisition flow
- Remember generated object

Same target rules as for RHS data access
Interpreting Shared-Key Encrypt.

\[
\frac{\Gamma; \Delta \vdash \mathbb{I} \{\dagger\}_k \gg \Delta''}{\Gamma; \Delta \vdash \mathbb{I} \{\dagger\}_k \gg \Delta''}
\]

\[
\forall A, B: \text{princ} \\
\forall k: \text{shK AB} \\
\forall \dagger: \text{msg}
\]

\[
\left\{ \begin{array}{l}
\mathcal{M}_I(\{\dagger\}_k), \mathcal{M}_I(k) \rightarrow \mathcal{M}_I(\dagger)
\end{array} \right\}
\]

Similar for public-key encryption and pairing

Intro.

MSR

DAS → DY

RHS data
RHS msg
New data
LHS data
LHS msg

Spec. → DAS

The Wolf Within
Now, what did we do?

LHS message interpretation

- Instantiate acting principal to I
- Meta-variables $\rightarrow$ Rule variables
- Types from auxiliary typing derivation
- Follow knowledge acquisition flow

- Conclusion $\rightarrow$ antecedent
- “Last” premises $\rightarrow$ consequent
Network Rules

\[
\frac{\Delta \vdash A \top \Rightarrow \Delta'}{\Gamma \vdash \Delta \parallel A N(t) \Rightarrow \Delta'}
\]

\[
\forall t: \text{msg}. N(t) \rightarrow M_{\text{I}}(t) \]

\[
\forall t: \text{msg}. M_{\text{I}}(t) \rightarrow N(t)
\]
... Other Rules?

Either

- redundant, or

\[
\left( \forall \tau : \text{msg. } N(\tau) \rightarrow N(\tau) \right)^I
\]

- or, innocuous (but sensible)

\[
\left( \forall \tau_1, \ldots, \tau_n : \text{msg. } M^{I}_{\gamma}(\tau_1, \ldots, \tau_n) \rightarrow M^{I}_{\gamma}(\tau_1), \ldots, M^{I}_{\gamma}(\tau_n) \right)^I
\]
Automating DAS Rule Design?

- One size does not fit all
- Look at protocol
  - Typed MSR spec.
  - Usage of constructs
- Involve construct declarations
  - Not sufficient
  - Use annotations
Generating DAS rules from use

- Interpret message components on LHS
- Access data (keys) on LHS
- Generate data on RHS
- Construct messages on RHS
- Access data on RHS
Accessing data

- Annotate the type of freely accessible data

  \[ \text{princ}: \ +\text{type} \]

- Make it conditional for dep. types

  \[ \text{pubK}: \ \ast\text{princ} \to \ +\text{type} \]
  \[ \text{privK}: \ \Pi\text{A}:\ast\text{princ}. \ +\text{pubK A} \to \ +\text{type} \]
Generating data

• Again, annotate types

nonce: !type

\[\text{shK: } +\text{princ} \rightarrow +\text{princ} \rightarrow !\text{type}\]

\[\text{shK: } +\text{princ} \rightarrow +\text{princ} \rightarrow !\text{type}\]
Pattern-matching constructors

- Mark arguments as input or output

\[
\_\_\_\_ : -\text{msg} \rightarrow -\text{msg} \rightarrow \text{msg}
\]

\[
\{\_\} : -\text{msg} \rightarrow \Pi A:+\text{princ.} \quad \Pi B:+\text{princ.} +\text{shK} \quad A \quad B \rightarrow \text{msg}
\]

\[
\{\{\_\}\} : -\text{msg} \rightarrow \Pi A:+\text{princ.} \quad \Pi k:+\text{pubK} \quad A.+\text{privK} \quad k \rightarrow \text{msg}
\]

\[\text{hash} : +\text{msg} \rightarrow \text{msg}\]

\[
[\_\_] : +\text{msg} \rightarrow \Pi A:*\text{princ.} \quad \Pi k:*\text{sigK} \quad A.+\text{verK} \quad k \rightarrow \text{msg}
\]

+ or -
Annotating Declarations

- Integrates semantics of types and constructors
- “Trimmed down” version of DAS
- Allows constructing DAS rules
  - ... and Dolev-Yao intruder
… alternatively

Compute DAS rules from protocol

- There are finitely many annotations
- Check protocol against each of them
- Keep the most restrictive ones that validate the protocol

Exponential!

More efficient algorithms?
The Wolf Within