Expressing Type-Flaw Attacks in a Strongly Typed Language

Iliano Cervesato iliano@itd.nrl.navy.mil

ITT Industries, Inc @ NRL - Washington DC

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

2nd International Workshop on Foundations for Secure/Survivable Systems and Networks

Tokyo, October 27th, 2001

Outline

WORK IN PROGRESS

- Type-confusion attacks
- Type-Flaw Attacks in MSR
- Simulation with Dolev-Yao Intruder

Type flaws
Example
Positions
Contribution
MSR 2.0
Simulation

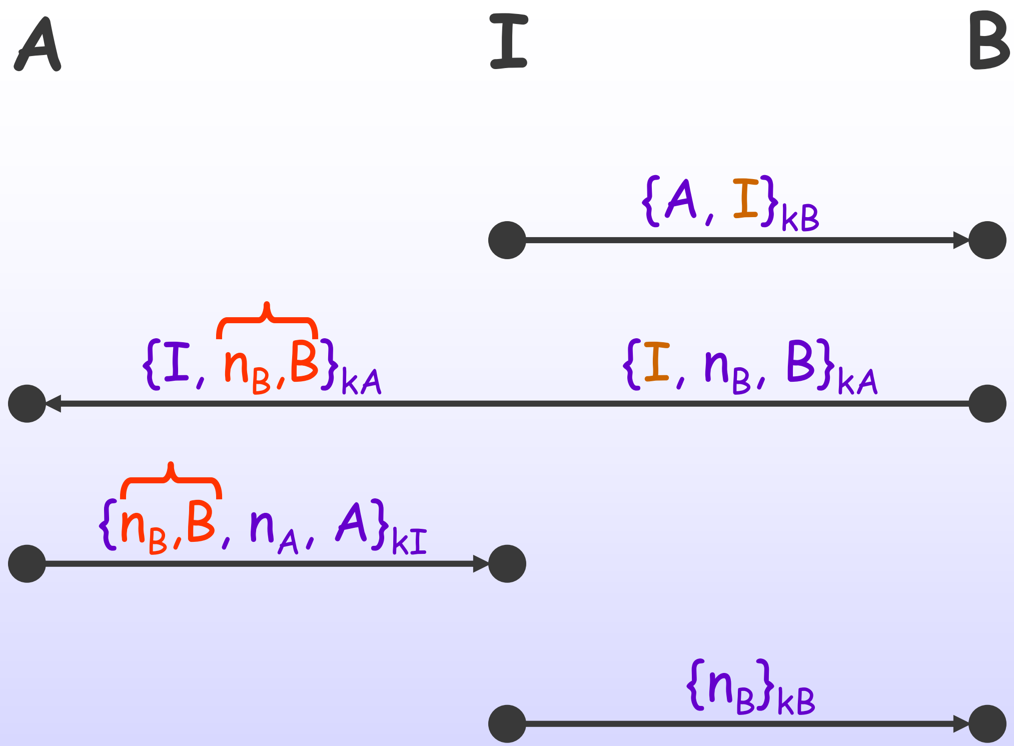
Type-Flaw Attacks

- Functionalities seen as "types"
  - Names
  - Nonces
  - Keys, …
- Violation
  - Principal misinterprets data
- Type flaw/confusion attack
  - Intruder manipulates message
  - Principal led to misuse data

Example: NSL

[Millen]

$A \to B: \{A, n_A\}_{kB}$
$B \to A: \{n_A, n_B, B\}_{kA}$
$A \to B: \{n_B\}_{kB}$



Confusion 1:
name/nonce

Confusion 2:
pair/nonce

B is fooled!

"Unlikely type violation"

Type flaws
- Example
- Positions
- Contribution

MSR 2.0
Simulation

Advocates

Type-flaw attacks are serious threats

- Push type-free specifications
  - Catch all “normal” attacks
  - … and type-confusion attacks too
  - Types are not real!

Opponents

Most type-flaw attacks are unrealistic

Type flaws
- Example
- Positions
- Contribution

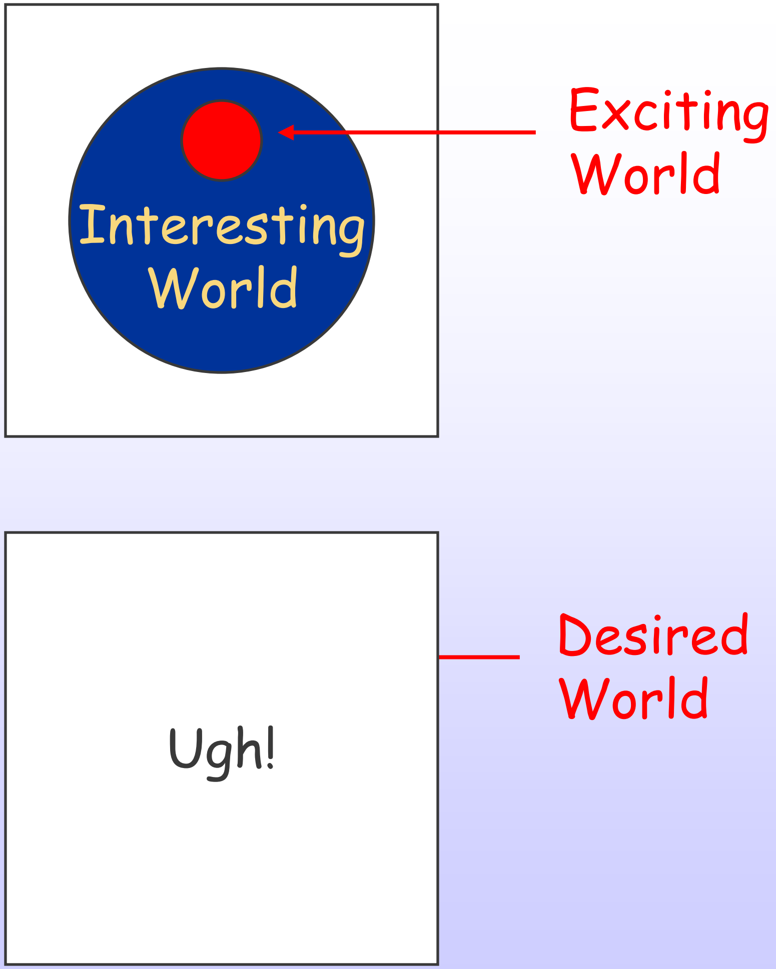
MSR 2.0

Simulation

- Push typed specification languages
  - Catch "real" attacks
  - Types guide search $\Rightarrow$ fast
  - Type-flaw attacks too low-level anyway

Prog. Languages vs. Security

- Types in programming languages
- Types in security



Type flaws

Example

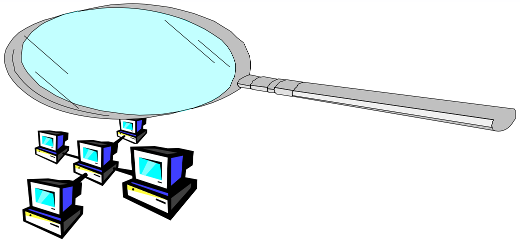
Positions

Contribution

MSR 2.0

Simulation

… in Reality



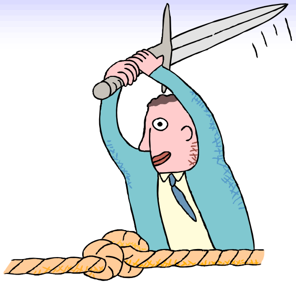
Type discriminants

- Data length
- Redundancy
- Explicit checks

- Resolve many situations …
- … but not all

"I so far found only one realistic type-flaw attack" [Meadows]

Contribution



- Reconcile
  - Typed languages
  - Type violations
- User specifies confusable types
  - Flexible
  - Abstract
- Support efficient simulation

Type flaws
MSR 2.0
Example
Typing
DAS
Execution
Intruder
Type flaws
Simulation

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
- Related to
  - strands
  - CIL
  - spi-calculus, …

What's in MSR 2.0 ?

- Multiset rewriting with existentials
- Dependent types w/ subsorting New
- Memory predicates New
- Constraints New

Type flaws

MSR 2.0

Example

Typing

DAS

Execution

Intruder

Type flaws

Simulation

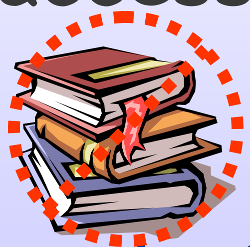
The Dolev-Yao Model of Security

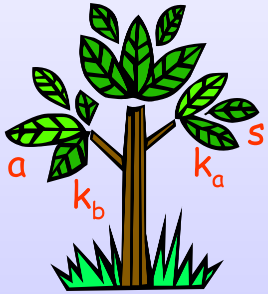
- Symbolic data
  - No bits

k_a

- Black-box cryptography
  - No guessing of keys

- Partially abstract data access
  - Knowledge soup





➢ Found in most protocol analysis tools
  - Tractability

Type flaws
MSR 2.0
Example
Typing
DAS
Execution
Intruder
Type flaws
Simulation

Roles

Role state pred. var. declarations

- Generic roles

$$\begin{pmatrix} \exists L: \tau'_1{}^{(x_1)} \times \dots \times \tau'_n{}^{(x_n)} & & & & \\ \dots & & & & \\ \forall \mathbf{x}:\tau. & lhs & \to & \exists \mathbf{y}:\tau'. & rhs \\ \dots & & \dots & & \dots \\ \forall \mathbf{x}:\tau. & lhs & \to & \exists \mathbf{y}:\tau'. & rhs \end{pmatrix}^{\forall A}$$

Role owner

- Anchored roles

$$\begin{bmatrix} \exists L: \tau'_1{}^{(x_1)} \times \dots \times \tau'_n{}^{(x_n)} & & & & \\ \dots & & & & \\ \forall \mathbf{x}:\tau. & lhs & \to & \exists \mathbf{y}:\tau'. & rhs \\ \dots & & \dots & & \dots \\ \forall \mathbf{x}:\tau. & lhs & \to & \exists \mathbf{y}:\tau'. & rhs \end{bmatrix}^{A}$$

Type flaws

MSR 2.0
- Example
- Typing
- DAS
- Execution
- Intruder
- Type flaws

Simulation

Rules

$$\forall x_1 : \tau_1. \; \ldots \; \forall x_n : \tau_n. \quad \text{lhs} \;\to\; \exists y_1 : \tau'_1. \; \ldots \; \exists y_{n'} : \tau'_{n'}. \quad \text{rhs}$$

lhs:

- $N(t)$ Network
- $L(t, \ldots, t)$ Local state
- $M_A(t, \ldots, t)$ Memory
- χ Constraints

rhs:

- $N(t)$ Network
- $L(t, \ldots, t)$ Local state
- $M_A(t, \ldots, t)$ Memory

NSL Initiator

$A \to B: \{A, n_A\}_{kB}$

$B \to A: \{n_A, n_B, B\}_{kA}$

$A \to B: \{n_B\}_{kB}$

- Type flaws
- MSR 2.0
  - Example
  - Typing
  - DAS
  - Execution
  - Intruder
  - Type flaws
- Simulation

$$\left(\begin{array}{lcccl}
\multicolumn{5}{l}{\exists L: \mathrm{princ} \times \mathrm{princ}^{(B)} \times \mathrm{pubK}\ B \times \mathrm{nonce}.} \\
\begin{array}{l}\forall B: \mathrm{princ} \\ \forall k_B: \mathrm{pubK}\ B\end{array} & \bullet & \to & \exists n_A{:}\mathrm{nonce}. & \begin{array}{l}L(A,B,k_B,n_A) \\ N(\{A, n_A\}_{kB})\end{array} \\
\begin{array}{l}\forall \ldots \\ \forall k_A: \mathrm{pubK}\ A \\ \forall\ k'_A: \mathrm{privK}\ k_A \\ \forall n_A, n_B: \mathrm{nonce}\end{array} & \begin{array}{l}L(A,B,k_B,n_A) \\ N(\{n_A, n_B, B\}_{kA})\end{array} & \to & & N(\{n_B\}_{kB})
\end{array}\right)^{\forall A}$$

Type flaws

MSR 2.0

- Example
- Typing
- DAS
- Execution
- Intruder
- Type flaws

Simulation

NSL Responder

$A \to B: \{A, n_A\}_{kB}$

$B \to A: \{n_A, n_B, B\}_{kA}$

$A \to B: \{n_B\}_{kB}$

$$\left(\begin{array}{lll}
\multicolumn{3}{l}{\exists L: \mathrm{princ}^{(B)} \times \mathrm{pubK}\ B^{(kB)} \times \mathrm{privK}\ k_B \times \mathrm{nonce}.} \\
\begin{array}{l} \forall k_B: \mathrm{pubK}\ B \\ \forall k'_B: \mathrm{privK}\ k_B \\ \forall A: \mathrm{princ} \\ \forall n_A: \mathrm{nonce} \\ \forall k_A: \mathrm{pubK}\ A \end{array} & N(\{A, n_A\}_{kB}) \to \exists n_B{:}\mathrm{nonce}. & \begin{array}{l} L(B, k_B, k'_B, n_B) \\ N(\{n_A, n_B, B\}_{kA}) \end{array} \\
\begin{array}{l} \forall \ \ldots \\ \forall n_B: \mathrm{nonce} \end{array} & \begin{array}{l} L(B, k_B, k'_B, n_B) \\ N(\{n_B\}_{kB}) \end{array} \to & \bullet
\end{array}\right)^{\forall B}$$

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

Subtyping

princ	::	msg
nonce	::	msg
pubK *A*	::	msg

- Allows atomic terms in messages
- Definable
  - Non-transmittable terms
  - Sub-hierarchies

Type Checking

New

$\Sigma \mid\!- P$

$\Gamma \mid\!- t : \tau$

t has type τ in Γ

P is well-typed in Σ

- Catches:
  - Encryption with a nonce
  - Transmission of a long term key
  - Circular key hierarchies, …
- Static and dynamic uses
- Decidable

Type flaws

MSR 2.0

- Example
- Typing
- DAS
- Execution
- Intruder
- Type flaws

Simulation

Data Access Specification

New

r is DAS-valid for A in Γ

$\Sigma \Vdash P$

$\Gamma \Vdash_A r$

P is DAS-valid in Σ

- Catches
  - A signing/encrypting with B's key
  - A accessing B's private data, …
- Static & Decidable
- Gives meaning to Dolev-Yao intruder
  - Completeness
  - Reconstructibility

Configurations

Active role set

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

State

- $N(t)$
- $L_l(t, \ldots, t)$
- $M_A(t, \ldots, t)$

Signature

- $a : \tau$
- $L_l : \underline{\tau}$
- $M\_ : \underline{\tau}$

Type flaws

MSR 2.0

- Example
- Typing
- DAS
- Execution
- Intruder
- Type flaws

Simulation

Execution Model

1-step firing

$$P \triangleright C \to C'$$

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

Variable Instantiation

Type flaws
MSR 2.0
Example
Typing
DAS
Execution
Intruder
Type flaws
Simulation

$$\frac{\Sigma \vdash t : \tau}{[S]^{R\,(\forall x{:}\tau.r,\rho)\ A}_{\Sigma} \rightarrow [S]^{R\,([t/x]r,\rho)\ A}_{\Sigma}}$$

Type checking guarantees proper usage

Rule Application

$$r = \quad F, \chi \rightarrow \exists \underline{n}:\underline{\tau}.\ G(\underline{n})$$

- Constraint check

$$\Sigma \models \chi \qquad \text{(constraint handler)}$$

- Firing

$$\underbrace{[S_1]}_{S,\ F}{}^{R(r,\rho)^A}{}_{\Sigma} \rightarrow \underbrace{[S_2]}_{S,\ G(\underline{c})}{}^{R\rho^A}{}_{\Sigma,\ \underline{c}:\underline{\tau}} \qquad \underline{c} \text{ not in } S_1$$

Type flaws

MSR 2.0

- Example
- Typing
- DAS
- Execution
- Intruder
- Type flaws

Simulation

Execution with an Attacker

$$P,\ P_I \triangleright C \rightarrow C'$$

- Selected principal(s): I
- Generic capabilities: P_I
  - Well-typed
  - DAS-valid
- Modeled completely within MSR

Expressing Type Violations ?

- Impossible !

$$\frac{\Sigma \vdash t : \tau}{[S]^{R\,(\forall x:\tau.r,\rho)}{}^{A}_{\Sigma} \;\rightarrow\; [S]^{R\,([t/x]r,\rho)}{}^{A}_{\Sigma}}$$

Typing forces principal to play by the rules

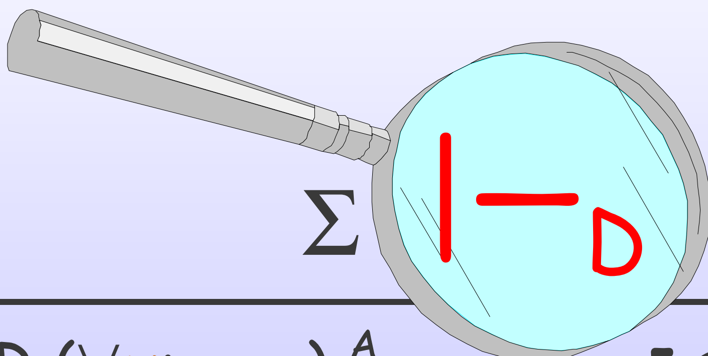
Expressing Type Violations !

Distinguish

- Static type-checking
- Dynamic type-checking

How things should be on paper

How things are in realty



$$\frac{\Sigma \vdash_D \tau}{[S]^R\,(\forall x{:}\tau.r,\rho)\,{}^A{}_\Sigma \;\rightarrow\; [S]^R\,([t/x]r,\rho)\,{}^A{}_\Sigma}$$

Subtyping Revisited

- Most rules have a rigid format

$$\frac{}{\Gamma, a{:}\tau, \Gamma' \mid\!- \; a : \tau}$$

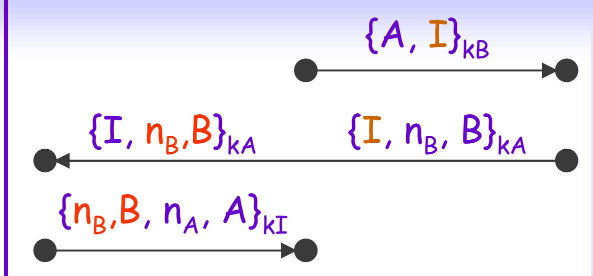
- Subtyping provides hook

$$\frac{\tau' :: \tau \qquad \Gamma \mid\!- \; t : \tau'}{\Gamma \mid\!- \; t : \tau}$$

Extend subtyping with confusable types

Type flaws
MSR 2.0
Example
Typing
DAS
Execution
Intruder
Type flaws
Simulation

A First Solution



princ	::	msg	static
nonce	::	msg	
pubK A	::	msg	
princ	::	nonce	dynamic extension
msg	::	nonce	

- Works …
- … but very raw
  - not every msg mistaken as a nonce
  - unwanted recursion

Towards a Polymorphic MSR

princ	:: msg	nonce$^+$	:: msg
nonce	:: msg	princ	:: nonce$^+$
pubK A	:: msg	nonce	:: nonce$^+$
pair α β	:: msg	pair princ nonce	:: nonce$^+$

pair : type -> type -> type.
_,_ : α -> β -> pair α β.

Confusable nonces

- Fine grained
- Captures what we want
- Recursion is up to us

Type flaws

MSR 2.0

- Example
- Typing
- DAS
- Execution
- Intruder
- Type flaws

Simulation

Summary

- Type violation (attacks) expressible in MSR
- Simple
- Flexible
  - You decide confusable types
  - Shades of gray in black/white positions

Types are good

Type flaws
MSR 2.0
Simulation
DY Intruder
Big steps
Type flaws

Simulation

- No attacker
  - Prototype
- With attacker
  - Verification
    - Model checking
    - Theorem proving
    - Process equivalence

The Dolev-Yao Intruder

Type flaws
MSR 2.0
Simulation
- DY Intruder
- Big steps
- Type flaws

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

- Found in most protocol analysis tools

- Modeled completely within MSR
- Generated automatically (mostly)

Intruder Simulation Approaches

- Take protocol text into account?
  - Blind / Focused
- Size of intruder steps
  - Small / Big
- Intruder representation
  - Explicit / Implicit

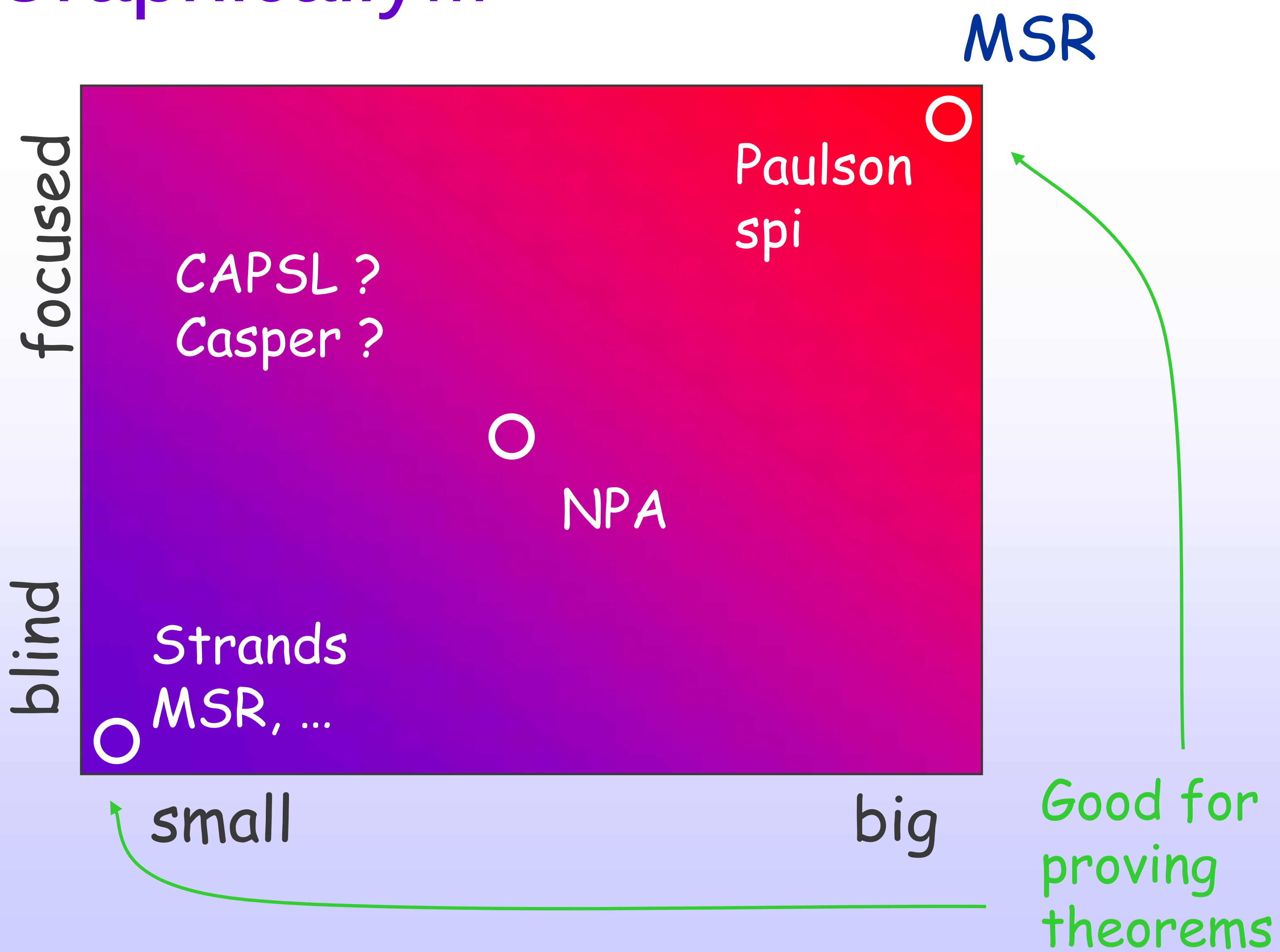
Type flaws

MSR 2.0

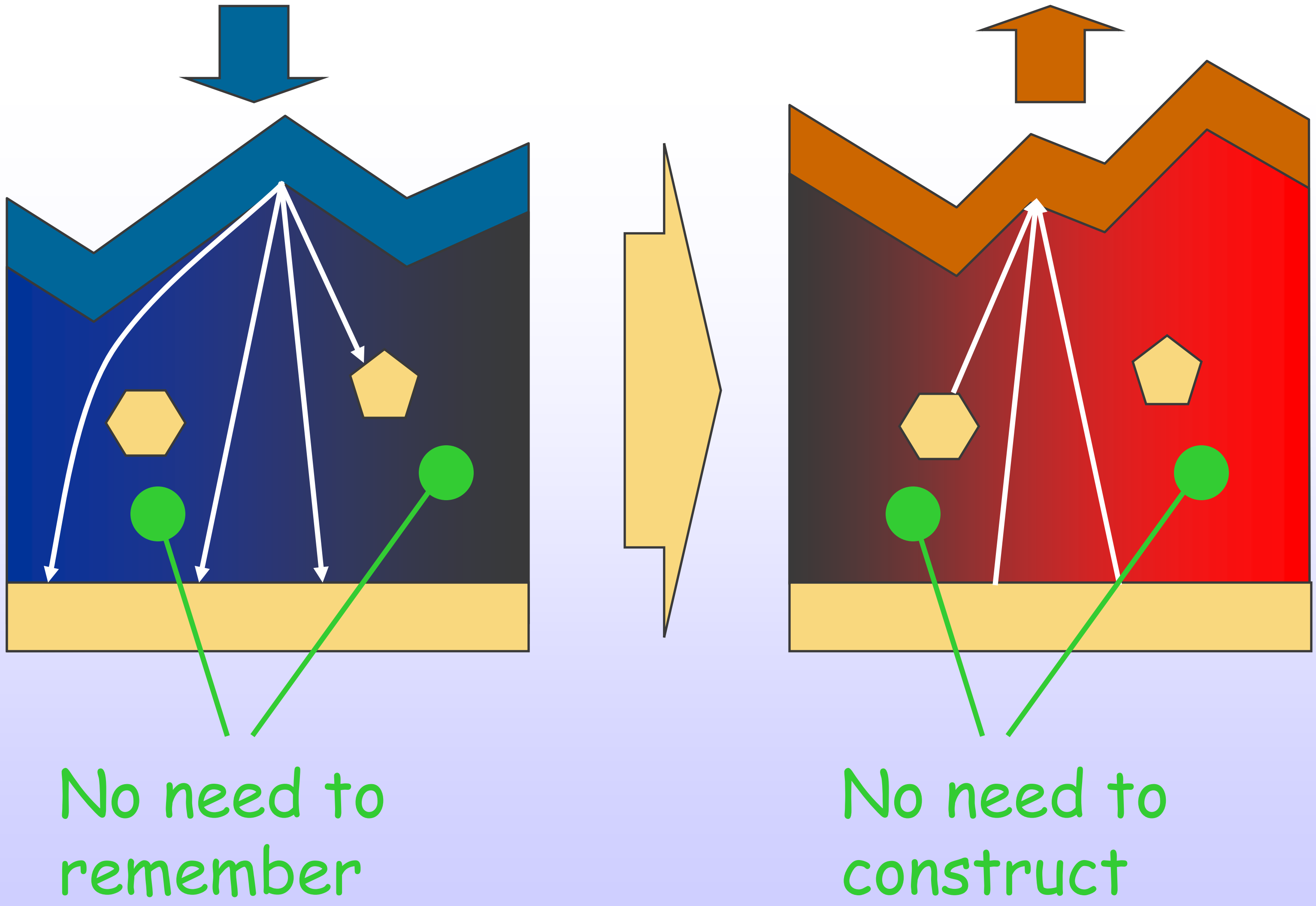
Simulation

- DY Intruder
- Big steps
- Type flaws

Graphically...



Intruder Activity



Type flaws
MSR 2.0
Simulation
- DY Intruder
- Big steps
- Type flaws

Intruder Activity Comparison

Disassembly

- Blind
  - Take pieces apart until
    - Atomic
    - Key unavailable
- Focused
  - Anticipate message contents
  - Memorize only what is needed

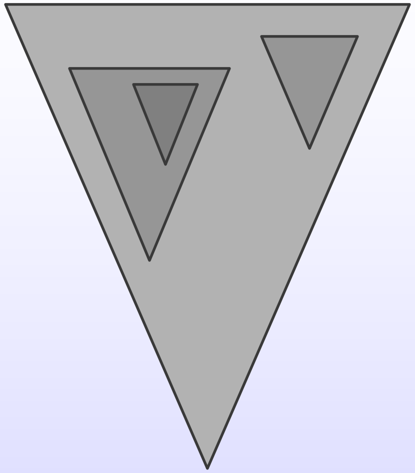
Assembly

- Blind
  - Put pieces together until meaningful message is built
- Focused
  - Build only usable messages

Type flaws
MSR 2.0
Simulation
DY Intruder
Big steps
Type flaws

Big-Step Message Disassembly

- Take typing derivation of (incoming) messages
- Encryption defines regions
- 1 role for each message
  - 1 rule for each region
  - Interface rule



$\Gamma \mid\!- \; t : \tau$

NSL – 1st Message

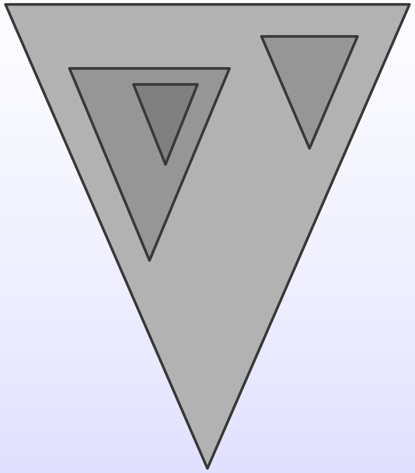
$A \to B: \{A, n_A\}_{kB}$
$B \to A: \{n_A, n_B, B\}_{kA}$
$A \to B: \{n_B\}_{kB}$

$$\left(\begin{array}{lll}
\multicolumn{3}{l}{\exists L: \text{princ} \times \text{msg}.} \\
\forall m: \text{msg} \quad N(m) & \to & L(I,m) \\
 & & M_I(m) \\
\forall A: \text{princ} \quad L(I, \{A, n_A\}_{kB}) & & M_I(A) \\
\forall k_B: \text{pubK } B \quad M_I(\{A, n_A\}_{kB}) & \to & M_I(n_A) \\
\forall k'_B: \text{privK } k_B \quad M_I(k'_B) & & M_I(k'_B) \\
\forall n_A: \text{nonce} & &
\end{array}\right)^I$$

- $M_I(m)$ "forgotten" as soon as k'_B is known
- Special case if k'_B known right away

Big-Step Message Assembly

- Take typing derivation of (outgoing) messages
- Encryption defines regions
- 1 role for each region
- Extras for generated data



$\Gamma \vdash t : \tau$

NSL – 1st Message

$A \to B: \{A, n_A\}_{kB}$

$B \to A: \{n_A, n_B, B\}_{kA}$

$A \to B: \{n_B\}_{kB}$

$$\left[\forall m: msg \quad M_I(m) \to N(m) \right]^I$$

$$\left[\begin{array}{ll} \forall A,B: princ & M_I(A) \\ \forall k_B: pubK\ B & M_I(n_A) \\ \forall n_A: nonce & M_I(k_B) \end{array} \to \begin{array}{l} N(\{A, n_A\}_{kB}) \\ M_I(A), M_I(n_A), M_I(k_B) \end{array} \right]^I$$

$$\left[\begin{array}{ll} \forall A,B: princ & M_I(A) \\ \forall k_B: pubK\ B & M_I(k_B) \end{array} \to \begin{array}{l} \exists n_A: nonce. \\ N(\{A, n_A\}_{kB}) \\ M_I(A), M_I(n_A), M_I(k_B) \end{array} \right]^I$$

What about confusable types ?

Creating Confusion

- Mark confusable objects
- Add rules for each option

$$\left(\begin{array}{llcl} \multicolumn{4}{l}{\exists \mathcal{L}: \mathrm{princ} \times \mathrm{nonce}^{+}.} \\ \begin{array}{l} \forall C: \mathrm{princ} \\ \forall n: \mathrm{nonce} \end{array} & \begin{array}{l} M_I(C) \\ M_I(n) \end{array} & \rightarrow & \begin{array}{l} \mathcal{L}(I, (C,n)) \\ M_I(C), M_I(n) \end{array} \\ \forall n: \mathrm{nonce} & M_I(n) & \rightarrow & \mathcal{L}(I, n), M_I(n) \\ \forall C: \mathrm{princ} & M_I(C) & \rightarrow & \mathcal{L}(I, C), M_I(C) \\ \begin{array}{l} \forall A,B: \mathrm{princ} \\ \forall k_B: \mathrm{pubK}\ B \\ \forall n: \mathrm{nonce}^{+} \end{array} & \begin{array}{l} \mathcal{L}(I,n) \\ M_I(A) \\ M_I(k_B) \end{array} & \rightarrow & \begin{array}{l} N(\{A, n\}_{kB}) \\ M_I(A), M_I(k_B) \end{array} \end{array} \right)^{I}$$

+ similar rule with $\exists n$:nonce

Type flaws
MSR 2.0
Simulation
- DY Intruder
- Big steps
- Type flaws

Making Sense of Confusion

$\exists L: \text{princ} \times \text{msg}.$ $\quad \exists L': \text{princ} \times \text{nonce}^{+}.$

$\forall m: \text{msg}$	$N(m)$	$\rightarrow$	$L(\mathbf{I}, m)$ $M_{\mathbf{I}}(m)$
$\forall A: \text{princ}$ $\forall k_B: \text{pubK } B$ $\forall k'_B: \text{privK } k_B$ $\forall n_A: \text{nonce}^{+}$	$L(\mathbf{I}, \{A, n_A\}_{kB})$ $M_{\mathbf{I}}(\{A, n_A\}_{kB})$ $M_{\mathbf{I}}(k'_B)$	$\rightarrow$	$M_{\mathbf{I}}(A)$ $L'(\mathbf{I}, n_A)$ $M_{\mathbf{I}}(k'_B)$
$\forall n: \text{nonce}$	$L'(\mathbf{I}, n)$	$\rightarrow$	$M_{\mathbf{I}}(n)$
$\forall A: \text{princ}$	$L'(\mathbf{I}, A)$	$\rightarrow$	$M_{\mathbf{I}}(A)$
$\forall A: \text{princ}$ $\forall n: \text{nonce}$	$L'(\mathbf{I}, (A, n))$	$\rightarrow$	$M_{\mathbf{I}}(A)$ $M_{\mathbf{I}}(n)$

$\mathbf{I}$

Further Optimizations

- Fold added rules in
  (unless confusion type is recursive)
  - Type-check in static type system
  - Bigger steps
- Simplify result using DAS rules
  - More compact
  - Formalizes "regions"
  - Automation

Type flaws
MSR 2.0
Simulation

Future Work

- Polymorphic MSR
- Strategies