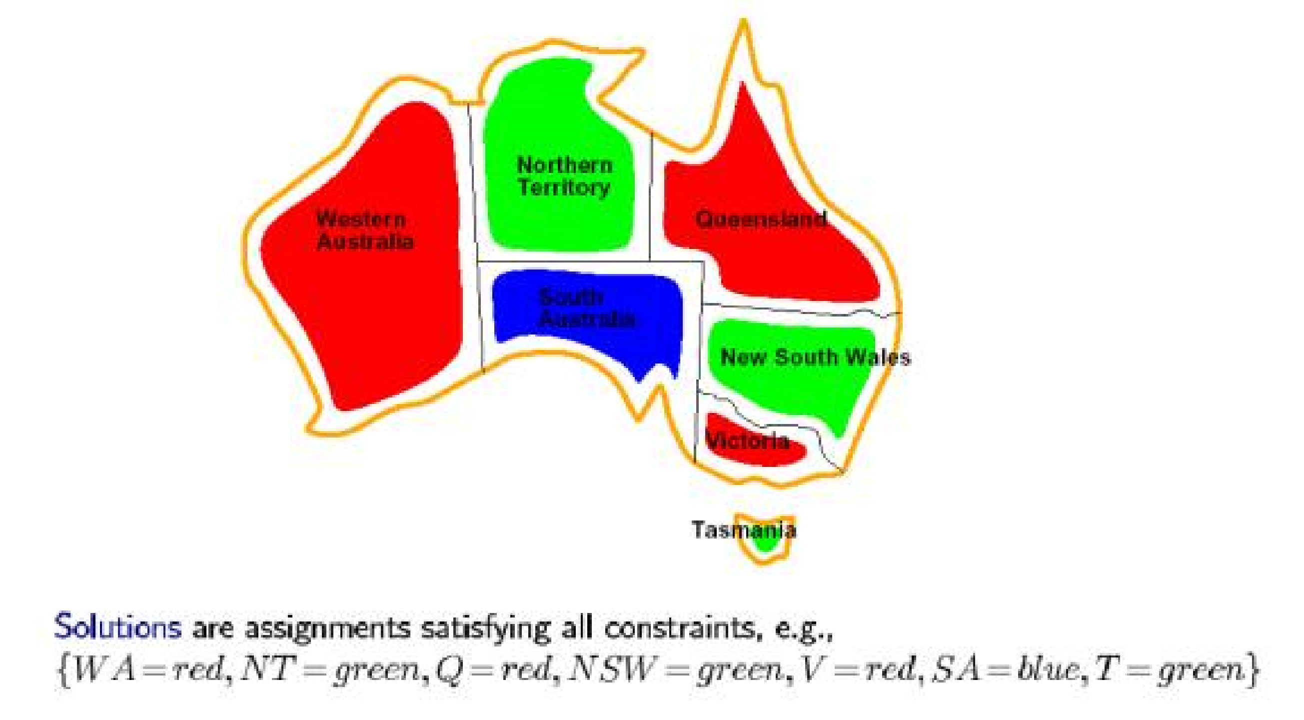
Constraint Solving for Protocol Analysis

- Papers by Jonathan Millen and Vitaly Shmatikov
  - “Constraint Solving for Bounded-Process Cryptographic Protocol Analysis”
- Presentation by James Wexler

Intro to Constraint Satisfaction Problems

- Set of variables that must be assigned values according to some constraints
- Constraints can constrain one variable or a set of variables
- Ex:



Intro to CSP continued

- Solution methods
  - Backtracking – basically depth first search of constraint set - used in this paper
  - Forward checking – eliminates impossible search states based on current state of solution
  - Constraint propagation – further eliminates impossible search states

Motivation of paper

- Secrecy expressible as reachability
  - Want to see if an undesirable state is reachable through use of the protocol with agents (including attacker)
- Reachability undecidable in general case
  - Can be decidable given enough restrictions on the problem
- The method will allow the analysis of protocols with key constructed from shared secrets (i.e. SSL)

Basics

- Protocol – represented by a form of strands that allows variables (parametric strands)
  - Variables allow one strand to represent all possible strands of a given role
- Attacker – Dolev-Yao attacker using term closure operator based on Paulson's synth and analz – doesn't use penetrator strands
- Analysis of protocol runs based on a bounded number of agents
- No typing

Term algebra

ε	The attacker or a principal compromised by the attacker (constant)	$h(t)$	Hash (modeled as a one-way function)
$[t_1, t_2]$	Pairing	$\lfloor t \rfloor_k^{\leftrightarrow}$	Term t encrypted with k using a symmetric algorithm. Keys are not required to be atomic terms.
$\mathrm{pk}(P)$	Messages encrypted with this public key can be decrypted by P using its corresponding private key. We assume that the private key of a public-private key pair is never transmitted as part of the protocol, or compromised in any way that might make it available as initial knowledge of the attacker. Therefore, the attacker can only decrypt terms encrypted with its own public key $\mathrm{pk}(\varepsilon)$.	$\lfloor t \rfloor_k$	Term t encrypted with k using a public-key algorithm. Any term can be used as if it were a public key.
		$\mathrm{sig}_k(t)$	Public-key signature of term t that is validated using key k. Since private keys of a key pair are never leaked, the attacker can only construct its own signatures $\mathrm{sig}_{\mathrm{pk}(\varepsilon)}(\ldots)$.

- Assume private keys are never leaked
- Only constructed keys using free term algebra (hashing), can't do xor

Example strands

- Needham-Schroeder-Lowe protocol
  - $$\mathit{Init}(A, B, N_A, N_B) = +[A, N_A]^{\rightarrow}_{\mathrm{pk}(B)} - [N_A, N_B, B]^{\rightarrow}_{\mathrm{pk}(A)} + [N_B]^{\rightarrow}_{\mathrm{pk}(B)}$$
  - Capital letters represent variables, lowercase letters will represent constants
  - Resp very similar – pluses and minuses reversed
  - Slightly different from original NSL
- A set of strands {init, resp} is called a semibundle
- Completable to bundle by supplying attacker computations and communication causality relation between messages sent and received

Attacker model

- Term set closure operation - F(T)
- A send node in a trace is realizable if it can be synthesized by the attacker from the set of messages sent in prior nodes
- Semibundle completable to bundle if it has a node ordering in which every send node is realizable.

Attacker Model

Analysis

$$
\begin{array}{llll}
\phi_{\mathrm{split}}(S) & = & S \cup x \cup y & \text{if } [x, y] \in S \\
\phi_{\mathrm{pdec}}(S) & = & S \cup x & \text{if } [x]_{\mathbf{pk}(\varepsilon)} \in S \\
\phi_{\mathrm{sdec}}(S) & = & S \cup x & \text{if } [x]_y^{\leftrightarrow}, y \in S
\end{array}
$$

Synthesis

$$
\begin{array}{llll}
\phi_{\mathrm{pair}}(S) & = & S \cup [x, y] & \text{if } x, y \in S \\
\phi_{\mathrm{penc}}(S) & = & S \cup [x]_y & \text{if } x, y \in S \\
\phi_{\mathrm{senc}}(S) & = & S \cup [x]_y^{\leftrightarrow} & \text{if } x, y \in S \\
\phi_{\mathrm{hash}}(S) & = & S \cup h(x) & \text{if } x \in S \\
\phi_{\mathrm{sig}}(S) & = & S \cup \mathrm{sig}_{\mathbf{pk}(\varepsilon)}(x) & \text{if } x \in S
\end{array}
$$

Encryption hiding

$$
\begin{array}{llll}
\phi_{\mathrm{open}}(S) & = & S \cup [x]_y^{\leftrightarrow} & \text{if } \lceil x \rceil_y \in S \\
\phi_{\mathrm{hide}}(S) & = & S \cup \lceil x \rceil_y & \text{if } [x]_y^{\leftrightarrow} \in S
\end{array}
$$

Attacker model

- Encryption hiding needed to support analysis of constructed keys.
- F(T) is a closure operation – idempotent, monotonic and extensive
- Same capabilities as penetrator strand approach but allows for easy conversion to constraint satisfaction problem
- Easily extendable

Goals of analysis

- Secrecy
  - Keeping N_B' secret in NSL case
  - Add one node strand (-N_B') to semibundle
  - Determine if semibundle is reachable
- Authentication
  - Add one node strand to semibundle containing message to be authenticated but no legit strand that sends it.
  - The paper is not clear on exactly what the strand would look like

Origination Assumption

- Variables always occur first time in any strand in a minus node
- Needed to prove completeness of decision procedure and helps us state and prove goals
- For principles, prefix a strand with received message containing the variables that would otherwise be sent first
  - In NSL add strand -[A, B] to the semibundle for initiator and responder identity variables

Constraint Generation

- Interleave strands in all of the possible ways and try to solve the constraint set
- One NSL merge:

$$
\begin{aligned}
&-[A,B] + [A,n_a]^{\rightarrow}_{\mathrm{pk}(B)} - [a,N_A]^{\rightarrow}_{\mathrm{pk}(b)} + [N_A,n_b,b]^{\rightarrow}_{\mathrm{pk}(a)} \cdots \\
&\quad \cdots - [n_a,N_B,B]^{\rightarrow}_{\mathrm{pk}(A)} + [N_B]^{\rightarrow}_{\mathrm{pk}(B)} - n_b
\end{aligned}
$$

- This semibundle had 2 responder strands
- Note the secret reception strand
- Exponential possible number of interleavings, optimization possible

Constraint set

- Constraints represent the messages the attacker would need to synthesize in order for the semibundle to be reachable (meaning the security property we are testing would be violated)

Constraint Set

- Constraints are of the form m:T
  - Each receive node is an m (message)
  - T is the last term set – terms originally known by the attacker and terms attacker has seen thus far in the protocol run (send nodes)
  - T_0 contains ground terms
- NSL example:

$$
\begin{aligned}
[A, B] &: T_0 = \{a, b, \varepsilon, \mathrm{pk}(\varepsilon)\} \\
[a, N_A]^{\rightarrow}_{\mathrm{pk}(b)} &: T_1 = T_0 \cup \{[A, n_a]^{\rightarrow}_{\mathrm{pk}(B)}\} \\
[n_a, N_B, B]^{\rightarrow}_{\mathrm{pk}(A)} &: T_2 = T_1 \cup \{[N_A, n_b, b]^{\rightarrow}_{\mathrm{pk}(a)}\} \\
n_b &: T_3 = T_2 \cup \{[N_B]^{\rightarrow}_{\mathrm{pk}(B)}\}
\end{aligned}
$$

- Constraint set is solvable if attacker can synthesize constraint messages from term set and F(T) operator

Solving the Constraint Set

- Reduction procedure
  - Applies rules that replace or eliminate a constraint
  - Terminates successfully when constrain set is a simple set – all left sides are simple variables
  - Reducible in many ways – creates a tree of possible solutions
  - If one path of tree terminates successfully then the semibundle is reachable

Reduction Procedure

C := initial constraint sequence

$\sigma := \emptyset$

`repeat`

    `let` $c^* = m : T$ be the first constraint in C

            s.t. m is not a variable

    `if` c^* not found

        `output` **Satisfiable!**

    apply rule *(elim)* to c^* until no longer applicable

    $\forall r \in R$

        `if` r is applicable to C

            $\langle C'; \sigma' \rangle := r(C; \sigma)$

            create node with C'; add $C \rightarrow C'$ edge

            `push` $\langle C'; \sigma' \rangle$

    $\langle C; \sigma \rangle :=$ `pop`

`until emptystack`

Figure 2: Reduction procedure **P**

Reduction Procedure

- Find first constraint that is not a variable
- Apply (elim) if possible
- Branch on all allowable reduction rules
- If path terminates in a satisfiable constraint set, it contains variable instantiations that the attacker has to make in order to stage a successful attack
- Sound and complete

Reduction rules

$$\frac{C_<, m : T, C_>; \sigma}{\tau C_<, \tau C_>; \tau \cup \sigma} \quad (un)$$

where $\tau = \mathrm{mgu}(m, t), t \in T$;

$$\frac{C_<, [m_1, m_2] : T, C_>; \sigma}{C_<, m_1 : T, m_2 : T, C_>; \sigma} \quad (pair)$$

$$\frac{C_<, h(m) : T, C_>; \sigma}{C_<, m : T, C_>; \sigma} \quad (hash)$$

$$\frac{C_<, [m]_k^{\rightarrow} : T, C_>; \sigma}{C_<, k : T, m : T, C_>; \sigma} \quad (penc)$$

$$\frac{C_<, [m]_k^{\leftrightarrow} : T, C_>; \sigma}{C_<, k : T, m : T, C_>; \sigma} \quad (senc)$$

$$\frac{C_<, \mathrm{sig}_{\mathrm{pk}(\varepsilon)}(m) : T, C_>; \sigma}{C_<, m : T, C_>; \sigma} \quad (sig)$$

$$\frac{C_<, m : [t_1, t_2] \cup T, C_>; \sigma}{C_<, m : t_1 \cup t_2 \cup T, C_>; \sigma} \quad (split)$$

$$\frac{C_<, m : [t]_{\mathrm{pk}(\varepsilon)}^{\rightarrow} \cup T, C_>; \sigma}{C_<, m : t \cup T, C_>; \sigma} \quad (pdec)$$

$$\frac{C_<, m : [t]_k^{\rightarrow} \cup T, C_>; \sigma}{\tau C_<, \tau m : \tau [t]_k^{\rightarrow} \cup \tau T, \tau C_>; \tau \cup \sigma} \quad (ksub)$$

where $\tau = \mathrm{mgu}(k, \mathrm{pk}(\varepsilon)), k \neq \mathrm{pk}(\varepsilon)$

$$\frac{C_<, m : T \cup [t]_k^{\leftrightarrow}, C_>; \sigma}{C_<, k : T \cup \lceil t \rceil_k, m : T \cup t \cup k, C_>; \sigma} \quad (sdec)$$

Note: $\lceil x \rceil_y$ unifies with $[x']_{y'}^{\leftrightarrow}$ *iff* $\exists \sigma$ s.t. $\sigma x = \sigma x', \sigma y = \sigma y'$

Example

- Interleaving:

$-[a, N_A]^{\rightarrow}_{\mathrm{pk}(b)}$	*to b*
$+[N_A, n_b, b]^{\rightarrow}_{\mathrm{pk}(a)}$	*from b to a*
$-[B, N_B]^{\rightarrow}_{\mathrm{pk}(A)}$	*to any A from any B*
$+[N_B, n_a, A]^{\rightarrow}_{\mathrm{pk}(B)}$	*from A to B*
$-n_b$	*secret reception*

The constraint set from this interleaving is:

$$
\begin{array}{lrcl}
1. & [a, N_A]^{\rightarrow}_{\mathrm{pk}(b)} & : & T_0 = \{a, b, \varepsilon, \mathrm{pk}(a), \mathrm{pk}(b)\} \\
2. & [B, N_B]^{\rightarrow}_{\mathrm{pk}(A)} & : & T_1 = T_0 \cup \{[N_A, [n_b, b]]^{\rightarrow}_{\mathrm{pk}(a)}\} \\
3. & n_b & : & T_1 \cup \{[N_B, [n_a, A]]^{\rightarrow}_{\mathrm{pk}(B)}\}
\end{array}
$$

We will follow one path leading to a solution. Note that we are treating concatenation as a binary right-associative operation. First, apply *(penc)* to (1):

$$
\begin{array}{lrcl}
1.1. & \mathrm{pk}(b) & : & T_0 \\
1.2. & [a, N_A] & : & T_0 \\
2. & [B, N_B]^{\rightarrow}_{\mathrm{pk}(A)} & : & T_1 = T_0 \cup \{[N_A, [n_b, b]]^{\rightarrow}_{\mathrm{pk}(a)}\} \\
3. & n_b & : & T_1 \cup \{[N_B, [n_a, A]]^{\rightarrow}_{\mathrm{pk}(B)}\}
\end{array}
$$

Example

Eliminate (1.1) with *(un)* and expand (1.2) with *(pair)*:

$$
\begin{array}{lrcl}
1.2.1. & a & : & T_0 \\
1.2.2. & N_A & : & T_0 \\
2. & [B, N_B]^{\rightarrow}_{\mathrm{pk}(A)} & : & T_1 = T_0 \cup \{[N_A, [n_b, b]]^{\rightarrow}_{\mathrm{pk}(a)}\} \\
3. & n_b & : & T_1 \cup \{[N_B, [n_a, A]]^{\rightarrow}_{\mathrm{pk}(B)}\}
\end{array}
$$

Eliminate (1.2.1) with *(un)* and skip (1.2.2) because it has a variable on the left. Apply *(un)* to (2) with the substitutions $B \mapsto N_A$, $N_B \mapsto [n_b, b]$ and $A \mapsto a$, eliminating (2).

$$
\begin{array}{lrcl}
1.2.2. & N_A & : & T_0 \\
3. & n_b & : & T_1 \cup \{[[n_b, b], [n_a, a]]^{\rightarrow}_{\mathrm{pk}(N_A)}\}
\end{array}
$$

Finally, apply *(ksub)* to (3) with $N_A \mapsto \varepsilon$. It should be clear after this that n_b will be exposed and the solution can be finished up easily. Installing the substitutions into the original semibundle yields the attack.

Example attack

- How the attack work - Type confusions:
  - Attacker name occupying a nonce field
  - [n_b, b] in first message occupying a nonce field
  - Only works if agent names are the same length as a nonce field and nonces can be two sizes (single and double length)
- Not very realistic but shows the power of this method of analysis

Negative opinions on paper

- Attacks shown in paper are not realistic
- Can't analyze encryption operations with associative and communitive properties such as xor, Diffie-Hellman exponentiation
- Paper doesn't show any real attacks on non-toy protocols.
- Only proves properties about running protocols with a fixed number of agents interacting

Positive opinions on paper

- Interesting use of AI concept to analyze protocols
- Can analyze some protocols with constructed keys
- Easily implementable – just a few pages of prolog
- Extendable to analyze unbounded processes – but will not terminate if attack not found

Questions??