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:

![Map of Australia](image-url)

*Solutions* are assignments satisfying all constraints, e.g.,
\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}
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
Assume private keys are never leaked.

- Only constructed keys using free term algebra (hashing), can't do xor
Example strands

- **Needham-Schroeder-Lowe protocol**
  
  \[
  \text{Init}(A, B, N_A, N_B) = 
  +[A, N_A]_{pk(B)} - [N_A, N_B, B]_{pk(A)} + [N_B]_{pk(B)}
  \]

  - 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

\[ \phi_{\text{split}}(S) = S \cup x \cup y \quad \text{if } [x, y] \in S \]
\[ \phi_{\text{pdec}}(S) = S \cup x \quad \text{if } [x]_{p^k(\varepsilon)} \in S \]
\[ \phi_{\text{sdec}}(S) = S \cup x \quad \text{if } [x]_{y}, y \in S \]

Analysis

\[ \phi_{\text{pair}}(S) = S \cup [x, y] \quad \text{if } x, y \in S \]
\[ \phi_{\text{penc}}(S) = S \cup [x]_{y} \quad \text{if } x, y \in S \]
\[ \phi_{\text{senc}}(S) = S \cup [x]_{y} \quad \text{if } x, y \in S \]
\[ \phi_{\text{hash}}(S) = S \cup h(x) \quad \text{if } x \in S \]
\[ \phi_{\text{sig}}(S) = S \cup \text{sig}_{p^k(\varepsilon)} \setminus x_i \quad \text{if } x \in S \]

Synthesis

Encryption hiding

\[ \phi_{\text{open}}(S) = S \cup [x]_{y} \quad \text{if } [x]_{y} \in S \]
\[ \phi_{\text{hide}}(S) = S \cup [x]_{y} \quad \text{if } [x]_{y} \in S \]
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:
  
  \[-[A, B] + [A, n_a]_{pk(B)} - [a, N_A]_{pk(b)} + [N_A, n_b, b]_{pk(a)} \ldots \]
  
  \[-[n_a, N_B, B]_{pk(A)} + [N_B]_{pk(B)} - n_b\]

- 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:

- 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

\[ C := \text{initial constraint sequence} \]
\[ \sigma := \emptyset \]

Repeat

Let \( c^* = m : T \) be the first constraint in \( C \)
\text{s.t.} \( m \) is not a variable

If \( c^* \) not found

Output \textbf{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 \( \langle C'; \sigma' \rangle \)

\[ \langle C; \sigma \rangle := \text{pop} \]

Until stack empty

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

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

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

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

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

\[
\frac{C_{<, [m_k]^r : T, C_>} ; \sigma}{C_{<, k : T, m : T, C_>} ; \sigma} \quad \text{(penc)}
\]

\[
\frac{C_{<, [m_k]^r : T, C_>} ; \sigma}{C_{<, k : T, m : T, C_>} ; \sigma} \quad \text{(senc)}
\]

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

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

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

\[
\frac{C_{<, m : [t]^r_k \cup T, C_>} ; \sigma}{C_{<, m : [t]^r_k \cup T, C_>} ; \sigma} \quad \text{(ksub)}
\]

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

\[
\frac{C_{<, m : T \cup [t]^r_k, C_>} ; \sigma}{C_{<, k : T \cup [t]^r_k, m : T \cup t \cup k, C_>} ; \sigma} \quad \text{(sdec)}
\]

Note: \([x]_y\) unifies with \([x']_y\) iff \( \exists \sigma \) s.t. \( \sigma x = \sigma x', \sigma y = \sigma y' \)
Example

- Interleaving:

\[
\begin{align*}
-a, N_A &\overset{\text{to } b}{\rightarrow} \text{pk}(b) \\
+ N_A, n_b, b &\overset{\text{from } b \text{ to } a}{\rightarrow} \text{pk}(a) \\
-B, N_B &\overset{\text{to any } A \text{ from any } B}{\rightarrow} \text{pk}(A) \\
+ N_B, n_a, A &\overset{\text{from } A \text{ to } B}{\rightarrow} \text{pk}(B) \\
-n_b &\text{secret reception}
\end{align*}
\]

The constraint set from this interleaving is:

1. \([a, N_A] \overset{\text{pk}(b)}{\rightarrow} : T_0 = \{a, b, \varepsilon, \text{pk}(a), \text{pk}(b)\} \)
2. \([B, N_B] \overset{\text{pk}(A)}{\rightarrow} : T_1 = T_0 \cup \{[N_A, [n_b, b]]_{\text{pk}(a)}\} \)
3. 

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):

1.1. \(\text{pk}(b) : T_0\)
1.2. \([a, N_A] : T_0\)
2. \([B, N_B] \overset{\text{pk}(A)}{\rightarrow} : T_1 = T_0 \cup \{[N_A, [n_b, b]]_{\text{pk}(a)}\} \)
3. 

Example

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

1.2.1. \[ a \rightarrow T_0 \]
1.2.2. \[ N_A \rightarrow T_0 \]
2. \[ [B, N_B]_{\text{pk}(A)} \rightarrow T_1 = T_0 \cup \{ [N_A, [n_b, b]]_{\text{pk}(a)} \} \]
3. \[ n_b \rightarrow T_1 \cup \{ [\gamma_B, [n_a, A]]_{\text{pk}(B)} \} \]

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 \( B \rightarrow N_A, N_B \rightarrow [n_b, b] \) and \( A \rightarrow a \), eliminating (2).

1.2.2. \[ \gamma_A \rightarrow T_0 \]
3. \[ n_b \rightarrow T_1 \cup \{ [n_b, b], [n_a, A]\} \]

Finally, apply (ksub) to (3) with \( N_A \rightarrow \varepsilon \). 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??