One Picture is Worth a Couple Dozen Connectives

Iliano Cervesato  iliano@itd.nrl.navy.mil

ITT Industries, inc @ NRL Washington, DC

http://theory.stanford.edu/~iliano

Joint work with Cathy Meadows

UMBC meeting  Baltimore, MD  October 1-2, 2003
How this work came about

Analysis of GDOI group protocol

- Requirements expressed in NPATRL
  - Novel group properties
  - Medium size specifications
    - Dozen operators
  - Lots of fine-tuning
- Difficult to read and share specs.
- Informal use of fault trees
  - Intuitive visualization medium
  - Became favored language
- Formal relation with NPATRL
Security Requirements

Describe what a protocol should do

• Verified by
  - Model checking
  - Mathematical proof
  - Pattern-matching (in some cases)

• Expressed
  - Informally
  - Semi-formally
  - Formal language

• Adequate for toy protocols
  BUT, do not scale to real protocols
Example: Kerberos 5

Theorem 1. For $C$ : client, $T$ : TGS, $C, T \neq 1$, $S$ : server, $k_C$ : $\text{dbK} C$, $k_T$ : $\text{dbK} T$, $\text{AKey} : \text{shK} C T$, and $n_2$ : nonce, if the beginning state of a finite trace does not contain $\text{I}(k_C), \text{I}(k_T)$, or any fact $F$ with $\rho_{k_T}(F; \text{AKey}, C) > 0$ or $\rho_{\text{AKey}}(F; C) > 0$, and at some point in the trace $T$ fires rule $\alpha_{4.1}$, consuming the fact $\text{N}([\text{AKey}, C]_{k_T}, [C]_{\text{AKey}}, C, S, n_2)$, then earlier in the trace, some $K : \text{KAS}$ fired rule $\alpha_{2.1}$, existentially generating $\text{AKey}$ and producing the fact $\text{N}([C, [\text{AKey}, C]_{k_T}, [\text{AKey}, n, T]_{k'}]$ for some $n$ : nonce and $k'$ : $\text{dbK} C$. Also, after $K$ fired this rule and before $T$ fired the rule in the hypothesis, $C$ fired rule $\alpha_{3.1}$ to create the fact $\text{N}(X, [C]_{\text{AKey}}, C, S', n')$ for some $X$ : msg, $S'$ : server, and $n'$ : nonce.

- Semi-formal
  - But very precise
- Bulky and unintuitive
  - Requires several readings to grasp
Example: GDOI

\[
\begin{align*}
\text{learn}(P, (), (K_G), \_)
\Rightarrow \quad & (\text{learn}(P, (), (K_G), \_) \land \text{gcks_createkey}(GCKS, (), (K_G), \_)) \\
& \land \text{gcks_createkey}(GCKS, (), (K'_G), \_))
\end{align*}
\]

\[
\begin{align*}
\lor \text{gcks_losegroupkey}(GCKS, (), (K_G), \_)
\lor & (\text{gcks_sendpushkey}(GCKS, (), (K_G, K'_G), N) \\
& \land \text{gcks_sendpullkey}(GCKS, M_d, (N_{GM}, K''_G, K_{GM}), \_)) \\
& \land \neg (\text{gcks_sendpushkey}(GCKS, (), (K_G, K'_G), N) \land \text{gcks_cancel}(GCKS, M_d, (N_{GM}), \_))
\end{align*}
\]

\[
\begin{align*}
\lor & \text{gcks_sendpullkey}(GCKS, M_d, (N_{GM}, K_G, K_{GM}), \_) \\
& \lor \text{gcks_losepairwisekey}(GCKS, (), (M, K_{GM}), \_) \\
& \lor \text{gcks_sendpullkey}(GCKS, M, (\_, K_G, K_{GM}), \_)
\end{align*}
\]

- **Formal**
  - NPATRL protocol spec. language
- **Ok for a computer**
- **Bulky and unintuitive for humans**
  - About 20 operators
Example: Authentication  [Lowe, CSFW’97]

Definition 1 (Aliveness). We say that a protocol guarantees to an initiator A aliveness of another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol.

Definition 2 (Weak Agreement). We say that a protocol guarantees to an initiator A weak agreement with another agent B if, whenever A (acting as an initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A.

Definition 3 (Non-Injective Agreement). We say that a protocol guarantees to an initiator A non-injective agreement with a responder B on a set of data items $t$ (where $t$ is a set of free variables appearing in the protocol description) if, whenever A (acting as an initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to the variables in $t$.

• Informal
  ➢ Made precise as CSP expressions

• Simple, but ...
  ➢ ... many very similar definitions
The Problem

• Desired properties are difficult to
  ➢ Phrase & get right
  ➢ Explain & understand
  ➢ Modify & keep right

• Examples
  ➢ Endless back and forth on GDOI
    ▪ Are specs. right now?
  ➢ K5 properties read over and over
Dealing with Textual Complexity

- HCI response: graphical presentation
- Our approach: **Dependence Trees**
  - Re-interpretation of fault trees
  - 2D representation of NPATRL
  - Intuitive for medium size specs.
Example: Kerberos 5

- Excises the gist of the theorem
- Highlights dependencies
- Fairly intuitive
  ➢ ... in a minute ...
Example: GDOI

- Isomorphic to NPATRL specifications
- Much more intuitive
  
  ... in a minute ...

Fault Tree Representation of Security Requirements
Example: Authentication

- Formalize definitions
- Easy to compare ...
  - ... and remember ...

Fault Tree Representation of Security Requirements
Rest of this Talk

• Logic for protocol specs
  - NPATRL Logic
  - NRL Protocol Analyzer fragment
  - Model checking

• Precedence trees
  - Fault trees
  - NPATRL semantics

• Analysis of an example

• Future Work
NPATRL

• Formal language for protocol requirements
  ➢ Simple temporal logic

• Designed for NRL Protocol Analyzer
  ➢ Simplify input of protocol specs
    ▪ Sequences of events that should not occur
  ➢ Applies beyond NPA

• Used for many protocols
  ➢ SET, GDOI, …
NPATRL Logic

Events

initiator_accept_key(A, (B,S), (K_{AB}, n_A), N)

Classical connectives: ∧, ∨, ¬, ...

“Previously”: # (Ø)

initiator_accept_key(A, (B,S), (K_{AB}, n_A), N) ⇒ # server_sent_key(S, (A,B), (K_{AB}), _)

Fault Tree Representation of Security Requirements
NPA Fragment

NPA uses a small fragment of NPATRL

\[ R ::= a \Rightarrow F \]
\[ F ::= E \mid \neg E \mid F_1 \land F_2 \mid F_1 \lor F_2 \]
\[ E ::= \#a \mid \#(a \land F) \]

• Efficient model checking
Fault Trees

• Safety analysis of system design
  - Root is a failure situation
    - Extended to behavior descriptions
  - Inner nodes are conditions enabling fault
    - Events
    - Combinators (logical gates)

• Example
  - A passenger needs a ticket and a photo ID to board a plane, but should not carry a weapon
Precedence Trees

- Fault tree representation of $\text{NPATRL}_{\text{NPA}}$
  - Isomorphism

\[
\begin{align*}
R & ::= a \Rightarrow F \\
F & ::= E \mid \neg E \mid F_1 \land F_2 \mid F_1 \lor F_2 \\
E & ::= \#a \mid \#(a \land F)
\end{align*}
\]
"Recency Freshness" in GDOI

if a member accepts a key from the controller in a protocol run, no newer key should have been distributed prior to the member's request

member_accept_key(M,G,(K_{GM},K_{old}),N) ⇒
   # gcks_loseparwisekey(G,(),(M,K_{GM}),_)
   ∨ ¬(# ( member_requestkey(M,G,(),N)
         ∧ #gcks_createkey(G,(),K_{new},K_{old}),_)))
“Sequential Freshness” in GDOI

if a member accepts a key from the group controller in a protocol run, then it should not have previously accepted a later key

member_accept_key(M, G, (K_{GM}, K_{old}), _) 
⇒
  # gcks_losepairwisekey(G, (), (M, K_{GM}), _)
  ∨ ¬(# (member_acceptkey(M, G, (K_{GM}, K_{new}), _))
    ∧ #(gcks_createkey(G, (), K_{new}, K'), _)
    ∧ #gcks_createkey(G, (), K_{old}, K''), _)))))
Conclusions

• Explored tree representation of protocol reqs.
  - Promising initial results
  - Complex requirements now intuitive

• Precedence trees
  - Draw from fault trees research
  - Specialized to NPATRL and NPA
  - NPATRL semantics
  - Better understanding of NPATRL

• Papers
  - “A Fault-Tree Representation of NPATRL Security Requirements”, with Cathy Meadows
    - WITS’03
    - TCS (long version, submitted)
Future Work – Theory

• What properties can be expressed?
  ➢ All of safety?
  ➢ Liveness?

• Graphical equivalence of requirements?

• Expressive power
  ➢ Recursive trees?
  ➢ More complex quantifier patterns?

• Graphical gist of theorems
  ➢ Useful classes?
  ➢ Proofs?
Future Work – Practice

- **Gain further experience**
  - Can they be used for other requirements?

- **Scaling up**
  - When are trees so big they are non-intuitive?
    - Existing requirements?
  - Modularity

- **Interaction with fault tree community**
  - Broader applications of dependence trees?
  - Tools we can use?
    - NPATRL <-> dependence trees