April 12: Expressiveness and Policy

- Expressiveness
  - Multiparent create
- Policies
- Trust
- Nature of Security Mechanisms
- Policy Expression Languages
- Limits on Secure and Precise Mechanisms
Expressive Power

• How do the sets of systems that models can describe compare?
  – If HRU equivalent to SPM, SPM provides more specific answer to safety question
  – If HRU describes more systems, SPM applies only to the systems it can describe
HRU vs. SPM

• SPM more abstract
  – Analyses focus on limits of model, not details of representation

• HRU allows revocation
  – SPM has no equivalent to delete, destroy

• HRU allows multiparent creates
  – SPM cannot express multiparent creates easily, and not at all if the parents are of different types because `can•create` allows for only one type of creator
Multiparent Create

• Solves mutual suspicion problem
  – Create proxy jointly, each gives it needed rights

• In HRU:

  \begin{verbatim}
  command multicreate(s_0, s_1, o)
  if r in a[s_0, s_1] and r in a[s_1, s_0]
  then
    create object o;
    enter r into a[s_0, o];
    enter r into a[s_1, o];
  end
  \end{verbatim}
SPM and Multiparent Create

- $cc$ extended in obvious way
  - $cc \subseteq TS \times \ldots \times TS \times T$

- Symbols
  - $X_1, \ldots, X_n$ parents, $Y$ created
  - $R_{1,i}, R_{2,i}, R_3, R_{4,i} \subseteq R$

- Rules
  - $cr_{P,i}(\tau(X_1), \ldots, \tau(X_n)) = Y/R_{1,1} \cup X_i/R_{2,i}$
  - $cr_{C}(\tau(X_1), \ldots, \tau(X_n)) = Y/R_3 \cup X_1/R_{4,1} \cup \ldots \cup X_n/R_{4,n}$
Example

• Anna, Bill must do something cooperatively
  – But they don’t trust each other

• Jointly create a proxy
  – Each gives proxy only necessary rights

• In ESPM:
  – Anna, Bill type $a$; proxy type $p$; right $x \in R$
  – $cc(a, a) = p$
  – $cr_{\text{Anna}}(a, a, p) = cr_{\text{Bill}}(a, a, p) = \emptyset$
  – $cr_{\text{proxy}}(a, a, p) = \{ \text{Anna}/x, \text{Bill}/\text{other } x \}$
2-Parent Joint Create Suffices

- Goal: emulate 3-parent joint create with 2-parent joint create
- Definition of 3-parent joint create (subjects \(P_1, P_2, P_3\); child \(C\)):
  - \(cc(\tau(P_1), \tau(P_2), \tau(P_3)) = Z \subseteq T\)
  - \(cr_{P_1}(\tau(P_1), \tau(P_2), \tau(P_3)) = C/R_{1,1} \cup P_1/R_{2,1}\)
  - \(cr_{P_2}(\tau(P_1), \tau(P_2), \tau(P_3)) = C/R_{2,1} \cup P_2/R_{2,2}\)
  - \(cr_{P_3}(\tau(P_1), \tau(P_2), \tau(P_3)) = C/R_{3,1} \cup P_3/R_{2,3}\)
General Approach

• Define agents for parents and child
  – Agents act as surrogates for parents
  – If create fails, parents have no extra rights
  – If create succeeds, parents, child have exactly same rights as in 3-parent creates
    • Only extra rights are to agents (which are never used again, and so these rights are irrelevant)
Entities and Types

- Parents $P_1, P_2, P_3$ have types $p_1, p_2, p_3$
- Child $C$ of type $c$
- Parent agents $A_1, A_2, A_3$ of types $a_1, a_2, a_3$
- Child agent $S$ of type $s$
- Type $t$ is parentage
  - if $X/t \in \text{dom}(Y)$, $X$ is $Y$’s parent
- Types $t, a_1, a_2, a_3, s$ are new types
can • create

- Following added to can • create:
  - $cc(p_1) = a_1$
  - $cc(p_2, a_1) = a_2$
  - $cc(p_3, a_2) = a_3$
    - Parents creating their agents; note agents have maximum of 2 parents
  - $cc(a_3) = s$
    - Agent of all parents creates agent of child
  - $cc(s) = c$
    - Agent of child creates child
Creation Rules

- Following added to create rule:
  - \( cr_P(p_1, a_1) = \emptyset \)
  - \( cr_C(p_1, a_1) = p_1/Rtc \)
    - Agent’s parent set to creating parent; agent has all rights over parent
  - \( cr_{P_{\text{first}}}(p_2, a_1, a_2) = \emptyset \)
  - \( cr_{P_{\text{second}}}(p_2, a_1, a_2) = \emptyset \)
  - \( cr_C(p_2, a_1, a_2) = p_2/Rtc \cup a_1/tc \)
    - Agent’s parent set to creating parent and agent; agent has all rights over parent (but not over agent)
Creation Rules

- \( cr_{p_{\text{first}}}(p_3, a_2, a_3) = \emptyset \)
- \( cr_{p_{\text{second}}}(p_3, a_2, a_3) = \emptyset \)
- \( cr_{C}(p_3, a_2, a_3) = p_3/Rtc \cup a_2/tc \)
  - Agent’s parent set to creating parent and agent; agent has all rights over parent (but not over agent)
- \( cr_{p}(a_3, s) = \emptyset \)
- \( cr_{C}(a_3, s) = a_3/tc \)
  - Child’s agent has third agent as parent \( cr_{p}(a_3, s) = \emptyset \)
- \( cr_{p}(s, c) = C/Rtc \)
- \( cr_{C}(s, c) = c/R_3 t \)
  - Child’s agent gets full rights over child; child gets \( R_3 \) rights over agent
Link Predicates

- Idea: no tickets to parents until child created
  - Done by requiring each agent to have its own parent rights
    - $\text{link}_1(A_2, A_1) = A_1/t \in \text{dom}(A_2) \land A_2/t \in \text{dom}(A_2)$
    - $\text{link}_1(A_3, A_2) = A_2/t \in \text{dom}(A_3) \land A_3/t \in \text{dom}(A_3)$
    - $\text{link}_2(S, A_3) = A_3/t \in \text{dom}(S) \land C/t \in \text{dom}(C)$
    - $\text{link}_3(A_1, C) = C/t \in \text{dom}(A_1)$
    - $\text{link}_3(A_2, C) = C/t \in \text{dom}(A_2)$
    - $\text{link}_3(A_3, C) = C/t \in \text{dom}(A_3)$
    - $\text{link}_4(A_1, P_1) = P_1/t \in \text{dom}(A_1) \land A_1/t \in \text{dom}(A_1)$
    - $\text{link}_4(A_2, P_2) = P_2/t \in \text{dom}(A_2) \land A_2/t \in \text{dom}(A_2)$
    - $\text{link}_4(A_3, P_3) = P_3/t \in \text{dom}(A_3) \land A_3/t \in \text{dom}(A_3)$
Filter Functions

• \( f_1(a_2, a_1) = a_1/t \cup c/Rtc \)
• \( f_1(a_3, a_2) = a_2/t \cup c/Rtc \)
• \( f_2(s, a_3) = a_3/t \cup c/Rtc \)
• \( f_3(a_1, c) = p_1/R_{4,1} \)
• \( f_3(a_2, c) = p_2/R_{4,2} \)
• \( f_3(a_3, c) = p_3/R_{4,3} \)
• \( f_4(a_1, p_1) = c/R_{1,1} \cup p_1/R_{2,1} \)
• \( f_4(a_2, p_2) = c/R_{1,2} \cup p_2/R_{2,2} \)
• \( f_4(a_3, p_3) = c/R_{1,3} \cup p_3/R_{2,3} \)
Construction

Create $A_1$, $A_2$, $A_3$, $S$, $C$; then

- $P_1$ has no relevant tickets
- $P_2$ has no relevant tickets
- $P_3$ has no relevant tickets
- $A_1$ has $P_1/Rtc$
- $A_2$ has $P_2/Rtc \cup A_1/tc$
- $A_3$ has $P_3/Rtc \cup A_2/tc$
- $S$ has $A_3/tc \cup C/Rtc$
- $C$ has $C/R_3t$
Construction

• Only \( \text{link}_2(S, A_3) \) true ⇒ apply \( f_2 \)
  – \( A_3 \) has \( P_3/Rtc \cup A_2/t \cup A_3/t \cup C/Rtc \)

• Now \( \text{link}_1(A_3, A_2) \) true ⇒ apply \( f_1 \)
  – \( A_2 \) has \( P_2/Rtc \cup A_1/tc \cup A_2/t \cup C/Rtc \)

• Now \( \text{link}_1(A_2, A_1) \) true ⇒ apply \( f_1 \)
  – \( A_1 \) has \( P_2/Rtc \cup A_1/t \cup C/Rtc \)

• Now all \( \text{link}_3s \) true ⇒ apply \( f_3 \)
  – \( C \) has \( C/R_3 \cup P_1/R_{4,1} \cup P_2/R_{4,2} \cup P_3/R_{4,3} \)
Finish Construction

• Now \( \text{link}_4 \) is true \( \Rightarrow \) apply \( f_4 \)
  
  – \( P_1 \) has \( C/R_{1,1} \cup P_1/R_{2,1} \)
  
  – \( P_2 \) has \( C/R_{1,2} \cup P_2/R_{2,2} \)
  
  – \( P_3 \) has \( C/R_{1,3} \cup P_3/R_{2,3} \)

• 3-parent joint create gives same rights to \( P_1, P_2, P_3, C \)

• If create of \( C \) fails, \( \text{link}_2 \) fails, so construction fails
Theorem

- The two-parent joint creation operation can implement an $n$-parent joint creation operation with a fixed number of additional types and rights, and augmentations to the link predicates and filter functions.
- **Proof**: by construction, as above
  - Difference is that the two systems need not start at the same initial state
Theorems

- Monotonic ESPM and the monotonic HRU model are equivalent.
- Safety question in ESPM also decidable if acyclic attenuating scheme
  - Proof similar to that for SPM
Expressiveness

- Graph-based representation to compare models
- Graph
  - Vertex: represents entity, has static type
  - Edge: represents right, has static type
- Graph rewriting rules:
  - Initial state operations create graph in a particular state
  - Node creation operations add nodes, incoming edges
  - Edge adding operations add new edges between existing vertices
Example: 3-Parent Joint Creation

- Simulate with 2-parent
  - Nodes $P_1, P_2, P_3$ parents
  - Create node $C$ with type $c$ with edges of type $e$
  - Add node $A_1$ of type $a$ and edge from $P_1$ to $A_1$ of type $e'$
Next Step

• $A_1, P_2$ create $A_2$; $A_2, P_3$ create $A_3$
• Type of nodes, edges are $a$ and $e'$

![Diagram](attachment:image.png)
Next Step

• $A_3$ creates $S$, of type $a$
• $S$ creates $C$, of type $c$
Last Step

- Edge adding operations:
  - $P_1 \rightarrow A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow S \rightarrow C$: $P_1$ to $C$ edge type $e$
  - $P_2 \rightarrow A_2 \rightarrow A_3 \rightarrow S \rightarrow C$: $P_2$ to $C$ edge type $e$
  - $P_3 \rightarrow A_3 \rightarrow S \rightarrow C$: $P_3$ to $C$ edge type $e$
Definitions

• Scheme: graph representation as above
• Model: set of schemes
• Schemes $A, B$ correspond if graph for both is identical when all nodes with types not in $A$ and edges with types in $A$ are deleted
Example

- Above 2-parent joint creation simulation in scheme *TWO*
- Equivalent to 3-parent joint creation scheme *THREE* in which $P_1, P_2, P_3, C$ are of same type as in *TWO*, and edges from $P_1, P_2, P_3$ to $C$ are of type $e$, and no types $a$ and $e'$ exist in *TWO*
Simulation

Scheme A simulates scheme B iff

- every state $B$ can reach has a corresponding state in $A$ that $A$ can reach; and
- every state that $A$ can reach either corresponds to a state $B$ can reach, or has a successor state that corresponds to a state $B$ can reach

  - The last means that $A$ can have intermediate states not corresponding to states in $B$, like the intermediate ones in $TWO$ in the simulation of $THREE$
Expressive Power

- If there is a scheme in $MA$ that no scheme in $MB$ can simulate, $MB$ less expressive than $MA$
- If every scheme in $MA$ can be simulated by a scheme in $MB$, $MB$ as expressive as $MA$
- If $MA$ as expressive as $MB$ and vice versa, $MA$ and $MB$ equivalent
Example

• Scheme $A$ in model $M$
  – Nodes $X_1, X_2, X_3$
  – 2-parent joint create
  – 1 node type, 1 edge type
  – No edge adding operations
  – Initial state: $X_1, X_2, X_3$, no edges

• Scheme $B$ in model $N$
  – All same as $A$ except no 2-parent joint create
  – 1-parent create

• Which is more expressive?
Can $A$ Simulate $B$?

- Scheme $A$ simulates 1-parent create: have both parents be same node
  - Model $M$ as expressive as model $N$
Can $B$ Simulate $A$?

- Suppose $X_1, X_2$ jointly create $Y$ in $A$
  - Edges from $X_1, X_2$ to $Y$, no edge from $X_3$ to $Y$
- Can $B$ simulate this?
  - Without loss of generality, $X_1$ creates $Y$
  - Must have edge adding operation to add edge from $X_2$ to $Y$
  - One type of node, one type of edge, so operation can add edge between any 2 nodes
No

- All nodes in $A$ have even number of incoming edges
  - 2-parent create adds 2 incoming edges
- Edge adding operation in $B$ that can edge from $X_2$ to $C$ can add one from $X_3$ to $C$
  - $A$ cannot enter this state
  - $B$ cannot transition to a state in which $Y$ has even number of incoming edges
    - No remove rule
- So $B$ cannot simulate $A$; $N$ less expressive than $M$
Theorem

• Monotonic single-parent models are less expressive than monotonic multiparent models

• Proof by contradiction
  – Scheme $A$ is multiparent model
  – Scheme $B$ is single parent create
  – Claim: $B$ can simulate $A$, without assumption that they start in the same initial state
  • Note: example assumed same initial state
Outline of Proof

• $X_1, X_2$ nodes in $A$
  - They create $Y_1, Y_2, Y_3$ using multiparent create rule
  - $Y_1, Y_2$ create $Z$, again using multiparent create rule
  - *Note*: no edge from $Y_3$ to $Z$ can be added, as $A$ has no edge-adding operation
Outline of Proof

- $W, X_1, X_2$ nodes in $B$
  - $W$ creates $Y_1, Y_2, Y_3$ using single parent create rule, and adds edges for $X_1, X_2$ to all using edge adding rule
  - $Y_1$ creates $Z$, again using single parent create rule; now must add edge from $X_2$ to $Z$ to simulate $A$
  - Use same edge adding rule to add edge from $Y_3$ to $Z$: cannot duplicate this in scheme $A$!
Meaning

• Scheme $B$ cannot simulate scheme $A$, contradicting hypothesis
• ESPM more expressive than SPM
  – ESPM multiparent and monotonic
  – SPM monotonic but single parent
Typed Access Matrix Model

• Like ACM, but with set of types $T$
  – All subjects, objects have types
  – Set of types for subjects $TS$

• Protection state is $(S, O, \tau, A)$
  – $\tau: O \rightarrow T$ specifies type of each object
  – If $X$ subject, $\tau(X)$ in $TS$
  – If $X$ object, $\tau(X)$ in $T - TS$
Create Rules

- **Subject creation**
  - create subject $s$ of type $ts$
  - $s$ must not exist as subject or object when operation executed
  - $ts \in TS$

- **Object creation**
  - create object $o$ of type $to$
  - $o$ must not exist as subject or object when operation executed
  - $to \in T - TS$
Create Subject

• Precondition: $s \notin S$
• Primitive command: create subject $s$ of type $t$
• Postconditions:
  - $S' = S \cup \{s\}$, $O' = O \cup \{s\}$
  - $(\forall y \in O)[\tau'(y) = \tau(y)]$, $\tau'(s) = t$
  - $(\forall y \in O')[a'[s, y] = \emptyset]$, $(\forall x \in S')[a'[x, s] = \emptyset]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$
Create Object

- Precondition: $o \notin O$
- Primitive command: create object $o$ of type $t$
- Postconditions:
  - $S' = S$, $O' = O \cup \{ o \}$
  - $(\forall y \in O)[\tau'(y) = \tau(y)]$, $\tau'(o) = t$
  - $(\forall x \in S')[a'[x, o] = \emptyset]$
  - $(\forall x \in S)(\forall y \in O)[a'[x, y] = a[x, y]]$
Definitions

- MTAM Model: TAM model without *delete*, *destroy*
  - MTAM is Monotonic TAM
- $\alpha(x_1:t_1, \ldots, x_n:t_n)$ create command
  - $t_i$ child type in $\alpha$ if any of *create subject* $x_i$ *of type* $t_i$ or *create object* $x_i$ *of type* $t_i$ occur in $\alpha$
  - $t_i$ parent type otherwise
Cyclic Creates

command $cry•havoc(s_1 : u, s_2 : u, o_1 : v, o_2 : v, o_3 : w, o_4 : w)$

create subject $s_1$ of type $u$;
create object $o_1$ of type $v$;
create object $o_3$ of type $w$;

enter $r$ into $a[s_2, s_1]$;
enter $r$ into $a[s_2, o_2]$;
enter $r$ into $a[s_2, o_4]$

end
Creation Graph

- $u$, $v$, $w$ child types
- $u$, $v$, $w$ also parent types
- Graph: lines from parent types to child types
- This one has cycles
Acyclic Creates

**command** cry\textcdot havoc\((s_1 : u, s_2 : u, o_1 : v, o_3 : w)\)

- create object \(o_1\) of type \(v\);
- create object \(o_3\) of type \(w\);
- enter \(r\) into \(a[s_2, s_1]\);
- enter \(r\) into \(a[s_2, o_1]\);
- enter \(r\) into \(a[s_2, o_3]\)

end
Creation Graph

- \( u \), \( w \) child types
- \( u \) parent type
- Graph: lines from parent types to child types
- This one has no cycles
Theorems

- Safety decidable for systems with acyclic MTAM schemes
  - In fact, it’s *NP-hard*

- Safety for acyclic ternary MATM decidable in time polynomial in the size of initial ACM
  - “Ternary” means commands have no more than 3 parameters
  - Equivalent in expressive power to MTAM
Key Points

• Safety problem undecidable
• Limiting scope of systems can make problem decidable
• Types critical to safety problem’s analysis
Security Policy

• Policy partitions system states into:
  – Authorized (secure)
    • These are states the system can enter
  – Unauthorized (nonsecure)
    • If the system enters any of these states, it’s a security violation

• Secure system
  – Starts in authorized state
  – Never enters unauthorized state
Confidentiality

- $X$ set of entities, $I$ information
- $I$ has the *confidentiality* property with respect to $X$ if no $x \in X$ can obtain information from $I$
- $I$ can be disclosed to others
- Example:
  - $X$ set of students
  - $I$ final exam answer key
  - $I$ is confidential with respect to $X$ if students cannot obtain final exam answer key
Integrity

- $X$ set of entities, $I$ information
- $I$ has the \textit{integrity} property with respect to $X$ if all $x \in X$ trust information in $I$
- Types of integrity:
  - Trust $I$, its conveyance and protection (data integrity)
  - $I$ information about origin of something or an identity (origin integrity, authentication)
  - $I$ resource: means resource functions as it should (assurance)
Availability

• $X$ set of entities, $I$ resource
• $I$ has the *availability* property with respect to $X$ if all $x \in X$ can access $I$
• Types of availability:
  – Traditional: $x$ gets access or not
  – Quality of service: promised a level of access (for example, a specific level of bandwidth) and not meet it, even though some access is achieved