Safety Analysis Of Attribute Based Access Control Model (ABACα)

Tahmina Ahmed and Ravi Sandhu
What is Safety in an Access Control Model?

Definition 1. Safety Of an Access Control Model: Given a system Of an access control model and an initial state of the system a safety question asks whether a subject can have certain rights on an object after execution of a sequence of authorized commands.

Example For HRU
Safety Question in given a HRU system, an initial state \( (S^0, O^0, M^0, R) \) where \( S^0, O^0 \) are set of subjects, objects from initial state and \( M^0 \) access matrix of initial state ,\( R \) is the set of Rights, a subject \( s \in S^0 \), an object \( o \in O^0 \) and a right \( r \in R \) whether in any state \( \gamma r \in M^\gamma[s,o] \).
Fig. 1. *Unified ABAC model structure.*
U, S and O represent finite sets of *existing* users, subjects and objects respectively.

UA, SA and OA represent finite sets of user, subject and object attribute functions respectively. (Henceforth referred to as simply attributes.)

P represents a finite set of permissions.

For each \( att \) in \( UA \cup SA \cup OA \), \( \text{Range}(att) \) represents the attribute’s range, a finite set of \textit{atomic} values.

\text{SubCreator}: \( S \rightarrow U \). For each subject \text{SubCreator} gives its creator.

\text{attType}: \( UA \cup SA \cup OA \rightarrow \{ \text{set, atomic} \} \). Specifies attributes as set or atomic valued.

Each attribute function maps elements in U, S and O to atomic or set values.

\[
\forall ua \in UA. \, ua : U \rightarrow \begin{cases} 
\text{Range}(ua) & \text{if attType}(ua) = \text{atomic} \\
2^{\text{Range}(ua)} & \text{if attType}(ua) = \text{set} 
\end{cases}
\]

\[
\forall sa \in SA. \, sa : S \rightarrow \begin{cases} 
\text{Range}(sa) & \text{if attType}(sa) = \text{atomic} \\
2^{\text{Range}(sa)} & \text{if attType}(sa) = \text{set} 
\end{cases}
\]
Safety Question On ABACα

Let the Initial state Users subjects objects can be represented as $U^0$, $S^0$, $O^0$. And $\mathcal{P}(S)$ is the Power Set of set $S$.

**Definition 2. Specific Object Safety Question:** Given an object $o \in O^0$ an attribute $oa \in OA$ and a value $v$ where $v \in \text{Range}(oa)$ if $\text{attType}(oa)=\text{atomic}$ or $v \in \mathcal{P}(\text{Range}(oa))$ if $\text{attType}(oa)=\text{set}$ whether the system can reach in a state where $oa(o)=v$?

**Definition 3. Specific Subject Safety Question:** Given a subject $s \in S^0$ an attribute $sa \in SA$ and a value $v$ where $v \in \text{Range}(sa)$ if $\text{attType}(sa)=\text{atomic}$ or $v \in \mathcal{P}(\text{Range}(sa))$ if $\text{attType}(sa)=\text{set}$ whether the system can reach in a state where $sa(s)=v$?
• User Creation, Modification and Deletion are Administrative operations of and out of scope for this analysis.

• Our assumption is that in respect of safety analysis consideration of subject deletion won’t make the system more vulnerable comparing to the system which disregard deletion. So we haven’t consider Subject deletion in our analysis.
Our Analysis Scope

User (u_1) → Creates Subject (s_1) → Creates Object (o_1) → Creates Subject (s_2) → Creates Object (o_2)

User (u_2) → Modifies Subject (s_1) → Modifies Object (o_1) → Modifies Subject (s_2) → Modifies Object (o_2)
To answer this question we need to give a try to reduce $\text{ABAC}_\alpha$ to a standard undecidable problem. We have tried with the following two problems:

- Halting Problem of Turing machine.
- Post Correspondence Problem.
A general Turing machine with one dimensional single tape $M$ is a 6 tuple: $\{Q, \Sigma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}} \}$, where:

- $Q$ is a finite set of states,
- $\Sigma$ is a finite set, alphabet with $\text{blank}$
- $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$ is the transition function,
- $q_0, q_{\text{accept}}, q_{\text{reject}} \in Q$ are the start state, accept state, and reject state, respectively, where $q_{\text{accept}} \neq q_{\text{reject}}$
The movement of the head in the tape in described as below:

- \( \delta(q,x) = (p,y,L) \) in state \( q \) the tape head searching for the cell containing \( x \) and the head write \( y \) on that cell moves one cell to the left on the tape and the new state should be named as \( p \). If the tape head is at the left end no movement will occur. Left transition can be of two types:
  1. Left transition when head pointing to the left end as it is a one way tape no creation will occur resulting this transition. Only cell content and state will change
  2. Left transition when head not pointing to the left end

- \( \delta(q,x) = (p,y,R) \) same as above only moves right.

Right transition can be of two types:

  1. Right transition when head pointing to the right end new cell should be created to move the head right.
  2. Right transition when head not pointing to the right end.
Turing Machine

Infinite Tape

1 0 0 0 1 1 1 0

Read / Write Head

Control Unit

State: Y

http://turing.slc.edu/~jmarshall/courses/2002/fall/cs30/Lectures/week08/Computation.html
Extension of ABAC\(\alpha\) Require to Construct TM

To construct one way single tape Turing Machine with ABAC\(\alpha\)

- Infinite attribute range for subjects and objects attribute.

- Ability to modify linked attribute. For example \(oa_1 \in OA, \, oa_2 \in OA\) and Range \((oa) = 0\), Let \(oa_1(o_1) = o_2\). During modification of \(o_1\) the function can access \(oa_2(oa_1(o_1))\) which is similar to \(oa_2(o_2)\).

- Ability to modify subject attribute during object creation.

\(ABAC_\alpha\) doesn’t supports any of the above functionality so without these extensions it’s not possible to simulate the one way single tape Turing machine.

So other than the extension of \(ABAC_\alpha\) attribute range we cannot simulate PCP with \(ABAC_\alpha\).
Construction Of Turing Machine With ABACα (+)

1. When head is pointing to the right end cell
   a. Turing Machine Movement
      
      | a | b | x |
      |---|---|---|
      |   |   |   |

      | a | b | y | blank |
      |---|---|----|------|
      |   |   |    |      |

   b. Corresponding ABAC Alpha(+) Implementation
      
      $S_0$
      
      rightmostchild=$o_k$

      $O_1$
      
      state=null cell=a begin=true right=$o_2$ end=false left=null parent=$s_3$

      $O_2$
      
      state=null cell=b begin=false right=true end=false left=null parent=$s_3$

      $O_k$
      
      state=q cell=x begin=false right=true end=true left=..... parent=$s_3$

      $O_0$
      
      state=null cell=y begin=false right=true end=false left=..... parent=none

      $S_0$
      
      rightmostchild=none

   attribute values of objects and subject before the move

   attribute values of objects and subject after the move
Simulation of Right Move

2. When head is not pointing to the right end cell
   a. Turing Machine Movement

   | a | x | b |
   +---+---+---+
   |   |   | z |
   +---+---+---+
   |   |   |   |
   +---+---+---+

   | a | y | b |
   +---+---+---+
   |   |   | z |
   +---+---+---+
   |   |   |   |
   +---+---+---+

   b. Corresponding ABAC Alpha(+) implementation

   \[ s_0 \]
   \[ \text{rightmostchild}=o_k \]

   \[ u_0 \]

   \[ s_0 \]
   \[ \text{rightmostchild}=o_k \]

   \[ \text{attribute values of objects and subject} \]

   \[ \text{attribute values of objects and subject} \]
The input of the problem consists of two finite lists $\alpha_1, \ldots, \alpha_N$ and $\beta_1, \ldots, \beta_N$ of words over some alphabet $A$ having at least two symbols. A solution to this problem is a sequence of indices $(i_k)_{1 \leq k \leq K}$ with $K \geq 1$ and $1 \leq i_k \leq N$ for all $k$, such that

$$\alpha_{i_1} \ldots \alpha_{i_K} = \beta_{i_1} \ldots \beta_{i_K}.$$ 

The decision problem then is to decide whether such a solution exists or not.

A solution to this problem would be the sequence $(3, 2, 3, 1)$, because

$$\alpha_3 \alpha_2 \alpha_3 \alpha_1 = bba + ab + bba + a = bbaaabbbaa = bb + aa + bb + baa = \beta_3 \beta_2 \beta_3 \beta_1.$$ 

Furthermore, since $(3, 2, 3, 1)$ is a solution, so are all of its "repetitions", such as $(3, 2, 3, 1, 3, 3, 1, 3, 3, 1)$, etc.; that is, when a solution exists, there are infinitely many solutions of this repetitive kind.

However, if the two lists had consisted of only $\alpha_2, \alpha_3$ and $\beta_2, \beta_3$ from those sets, then there would have been no solution (the last letter of any such $\alpha$ string is not the same as the letter before it, whereas $\beta$ only constructs pairs of the same letter).

A convenient way to view an instance of a Post correspondence problem is as a collection of blocks of the form

$$\alpha_i, \beta_i$$

Source: http://en.wikipedia.org/wiki/Post_correspondence_problem
PCP Construction needs the

- Range of the attribute value should be infinite
- Its type should be string.

ABAC_\alpha does not support attributes with infinite range. Range of every attribute a \in UA \cup SA \cup OA is finite.
Is Safety Of ABAC Decidable?

For Convenience of Analysis we have defined some notation

Let $\mathcal{P}(S)$ denote the power set of $S$. Note that each element in $\mathcal{P}(S)$ is a set.

For each $ua$ in $UA$, $UASET_{ua} = \begin{cases} \{ua\} \times \text{Range}(ua) & \text{if attType}(ua) = \text{atomic} \\ \{ua\} \times 2^{\text{Range}(ua)} & \text{if attType}(ua) = \text{set} \end{cases}$

For each $sa$ in $SA$, $SASET_{sa} = \begin{cases} \{ua\} \times \text{Range}(sa) & \text{if attType}(sa) = \text{atomic} \\ \{ua\} \times 2^{\text{Range}(sa)} & \text{if attType}(sa) = \text{set} \end{cases}$

For each $oa$ in $OA$, $OASET_{oa} = \begin{cases} \{oa\} \times \text{Range}(oa) & \text{if attType}(oa) = \text{atomic} \\ \{oa\} \times 2^{\text{Range}(oa)} & \text{if attType}(oa) = \text{set} \end{cases}$

Let $POW_{ua} = \mathcal{P}(\bigcup_{ua \in UA} UASET_{ua})$

$UASET = \{x|x \in POW_{ua} \land |x| = |UA| \land \forall a \in UA. \forall val1, val2 \in \text{Range}(a). \langle a, val1 \rangle \in x \land (val1 \neq val2) \rightarrow \langle a, val2 \rangle \not\in x\}$

Let $POW_{oa} = \mathcal{P}(\bigcup_{oa \in OA} OASET_{oa})$

$OASET = \{x|x \in POW_{oa} \land |x| = |OA| \land \forall a \in OA. \forall val1, val2 \in \text{Range}(a). \langle a, val1 \rangle \in x \land (val1 \neq val2) \rightarrow \langle a, val2 \rangle \not\in x\}$
Example of UASET

let UA = \{ua_1, ua_2\}
Range(ua_1) = \{1, 2, 3\}
atattype(ua_1) = atomic
Range(ua_2) = \{a, b, c\}
atattype = set

\[
\text{UASET}_{ua_1} = \{1, 2, 3\} \\
\text{UASET}_{ua_2} = \{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\} \\
\text{UASET} = \{\langle 1, \{\}\rangle, \{1, \{a\}\}, \ldots \}
\]
**Notations**

\[ Q_{\text{users}}^\gamma = \text{AllPossibleStatesOfUsers}(U^\gamma, \text{UASET}) \]

\[ \text{AllPossibleStatesOfUsers}(U^\gamma, \text{UASET}) \]

\[ = \{x | x \in \mathcal{P}(U^\gamma \times \text{UASET}) \land |x| = |U^\gamma| \land \forall u \in U^\gamma. \forall \text{uaset}_1, \text{uaset}_2 \in \text{UASET}. \langle u, \text{uaset}_1 \rangle \in x \land (\text{uaset}_1 \neq \text{uaset}_2) \rightarrow \langle u, \text{uaset}_2 \rangle \notin x \} \]

\[ Q_{\text{objects}}^\gamma = \text{AllPossibleStatesOfObjects}(O^\gamma, \text{OASET}) \]

\[ \text{AllPossibleStatesOfObjects}(O^\gamma, \text{OASET}) \]

\[ = \{x | x \in \mathcal{P}(O^\gamma \times \text{OASET}) \land |x| = |O^\gamma| \land \forall o \in O^\gamma. \forall \text{oaset}_1, \text{oaset}_2 \in \text{OASET}. \langle o, \text{oaset}_1 \rangle \in x \land (\text{oaset}_1 \neq \text{oaset}_2) \rightarrow \langle o, \text{oaset}_2 \rangle \notin x \} \]

\[ Q_{\text{subjects}}^\gamma = \text{AllPossibleStatesOfSubjects}(S^\gamma, \text{SASET}) \] Where

\[ \text{AllPossibleStatesOfSubjects}(S^\gamma, \text{SASET}) \]

\[ = \{x | x \in \mathcal{P}(S^\gamma \times \text{SASET}) \land |x| = |S^\gamma| \land \forall s \in S^\gamma. \forall \text{saset}_1, \text{saset}_2 \in \text{SASET}. \langle s, \text{saset}_1 \rangle \in x \land (\text{saset}_1 \neq \text{saset}_2) \rightarrow \langle o, \text{saset}_2 \rangle \notin x \} \]
EXAMPLE FOR $Q^\gamma_{users}$
Let $U^\gamma = \{u_1, u_2, u_3\}$
$UASET = \{uaset_1, uaset_2\}$

$$Q^\gamma_{users} = \{ \langle\langle u_1, uaset_1, \rangle, \langle u_2, uaset_1, \rangle, \langle u_3, uaset_1, \rangle \rangle, \langle\langle u_1, uaset_1, \rangle, \langle u_2, uaset_2, \rangle, \langle u_3, uaset_1, \rangle \rangle, \langle\langle u_1, uaset_1, \rangle, \langle u_2, uaset_1, \rangle, \langle u_3, uaset_2, \rangle \rangle, \langle\langle u_1, uaset_2, \rangle, \langle u_2, uaset_2, \rangle, \langle u_3, uaset_2, \rangle \rangle, \langle\langle u_1, uaset_2, \rangle, \langle u_2, uaset_1, \rangle, \langle u_3, uaset_2, \rangle \rangle, \langle\langle u_1, uaset_2, \rangle, \langle u_2, uaset_1, \rangle, \langle u_3, uaset_1, \rangle \rangle \}$$
Notational Change From ABACα

- LConstrObj contains only the current and new attribute value of the subject and object so without loss of generality we can rewrite ConstrObj like as follows:
  \[ \text{ConstrObj} \left( \text{set}^{\text{curr}}: \text{SASET}, \text{set}^{\text{new}}: \text{OASET} \right) \]

- LConstrObjMod contains only the current and new attribute value of the subject and object so without loss of generality we can rewrite ConstrObj like as follows:
  \[ \text{ConstrObj} \left( \text{set}^{\text{curr}}: \text{SASET}, \text{set}^{\text{curr}}: \text{OASET}, \text{set}^{\text{new}}: \text{OASET} \right) \]

- LConstrSub contains only the current and new attribute value of the user and subject so without loss of generality we can rewrite ConstrSub like as follows:
  \[ \text{ConstrSub} \left( \text{set}^{\text{curr}}: \text{UASET}, \text{set}^{\text{new}}: \text{SASET} \right) \]

- LConstrSubMod contains only the current and new attribute value of the subject and object so without loss of generality we can rewrite ConstrObj like as follows:
  \[ \text{ConstrSubMod} \left( \text{set}^{\text{curr}}: \text{UASET}, \text{set}^{\text{curr}}: \text{SASET}, \text{set}^{\text{new}}: \text{SASET} \right) \]
Observation Towards Decidability Analysis

- In ABACα Object creation and modification of newly created object’s attribute doesn’t have any impact on existing subjects’ and objects’ attribute value. So we can disregard newly created object during our analysis.

- Subject Creation and modification of any subject attribute value has an impact on object creation and modification in the system.

- Only the subject’s attribute value not the subject itself is important for the analysis.

- We need to create as many subjects necessary and sufficient to figure out the worst case modification of object’s attribute value. This will be called the unfolding state.
Operations on Unfolding State

- No subject or object creation operation
- Construct a finite state machine with unfolding state subject and initial state objects
- Transition functions are
  
  Subject Modification: user can modify only initial state subjects
  
  Object Modification: Any Subject From the unfolding State can modify objects of initial state.
Subject Attribute Update Graph For \( ABAC_\alpha \) (\( SAUG_{ABAC_\alpha} \))

Nodes: \( \{ \langle x, y \rangle \mid x \in SASET \land y \in UASET \} \)

Edges: there is an edge from \( u \) to \( v \) where

\[
u = \langle p, r \rangle
\]

and

\[
v = \langle q, r \rangle
\]

and \( \text{ConstrSubMod}(r, p, q) = true \)
Subject Attribute Update Graph

\[ \text{ConstrSubMod}(uaset_1, saset_1, saset_2) = \text{true} \]

\[ \text{uaset}_1, \text{saset}_1 \]
\[ \text{uaset}_1, \text{saset}_2 \]
\[ \text{uaset}_1, \text{saset}_3 \]
\[ \text{uaset}_2, \text{saset}_3 \]
\[ \text{uaset}_2, \text{saset}_2 \]
\[ \text{uaset}_2, \text{saset}_4 \]

© Tahmina Ahmed  
World-Leading Research with Real-World Impact!  
25
Subject Attribute Creation set (SACS) SACS is a set of tuples \( \{ (u, saset) | \text{ where } u \in U^0 \land saset \in SASET \text{ ConstrObj}(uaset(u), saset) = true \} \)

Subject Attribute Reachable set (SARS) SARS is a set of tuples \( \{ (u, saset_2) | \text{ where there exists an } saset_1 \text{ such that } (u, saset_1) \in SACS \land saset_1 \neq saset_2 \land \text{ There is a PATH in } SAUG_{ABAC_\alpha} \text{ from } (uaset(u), saset_1) \text{ to } (uaset(u), saset_2) \} \)
Definition 4. The Subject Mapping Function $\sigma_{\text{mapping}}$ of $\text{ABAC}_\alpha$ scheme maps two subjects from any state $h$ to the unfolding state when both are derived from same initial state. More Precisely for $\text{ABAC}_\alpha$ system, if $h$ is any state and $k$ is the unfolding state construction of the system both are derived from the same initial state 0 then the

Subject Mapping Function $\sigma_{\text{mapping}}^{h,k} : S^h \rightarrow S^k$

That is $\sigma_{\text{mapping}}^{h,k}(X) \equiv \{ Y \in S^k \mid (X \in S^0 \implies X=Y) \lor (X \notin S^0 \implies (\text{saset}^h(X) = \text{saset}^k(Y) \land \text{SubCreator}(X) = \text{SubCreator}(Y) \land Y \notin S^0) \}$
Unfolding Algorithm

Let the set of Subjects being created during the construction of the unfolding state is $S^{USC}$.

**UnFolding Algorithm**

\[
i = 0 \\
S^{USC} = \emptyset \\
q_{subjects}^{USC} = \emptyset \\
\text{for each } \langle u, saset_1, saset_2 \rangle \in SARS \\
\quad S^{USC} = S^{USC} \cup s_i \\
\quad \text{for each } \langle sa, val \rangle \in saset_2 \\
\quad \quad sa^{US}(s_i) = val \\
\quad q_{subjects}^{USC} = q_{subjects}^{USC} \cup \langle s_i, saset_2 \rangle \\
\quad i++ \\
\]

\[
S^{US} = (S)^0 \cup S^{USC} \\
q_{subjects}^{US} = (q_{subjects})^0 \cup q_{subjects}^{USC} \\
\]

runtime for the construction of unfolding state: $U^0 \times SASET$

Where $S^0 =$ Set Of All subjects from initial state

$q_{subjects}^0 =$ set of all possible subject SASET tuple
RUNTIME ANALYSIS

Unfolding state Creation : (| U^0 | \times | SASET |)

Finite State Machine States :
((| S^0 | \times | SASET |) + | U^0 \times SASET |) \times (| O^0 | \times | OASET |)
Thank you!