Attribute-Aware Relationship-Based Access Control for Online Social Networks

Yuan Cheng, Jaehong Park and Ravi Sandhu
Institute for Cyber Security
University of Texas at San Antonio
7/14/2014

28th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec 2014)

Outline

- Relationship-based Access Control (ReBAC)
- Motivation
- $UURAC_A$ Model
- Algorithm
- Conclusion

Relationship-based Access Control

- Users in OSNs are connected by social relationships (user-to-user relationships)
- Owner of the resource can control its release based on such relationships between the access requester and the owner
- Access conditions are usually based on type, depth, or strength of relationships

Related Work

	Fong 2009	Fong 2011	Carminati 2009a	Carminati 2009b	$UURAC_A$
Relationship Category					
Multiple Relationship Types		√	√	√	√
Directional Relationship		√	√		√
Model Characteristics					
Policy Individualization	√	√	√	√	√
User & Resource as a Target				(partial)	√
Outgoing/Incoming Action Policy				(partial)	√
Relationship Composition					
Relationship Depth	0 to 2	0 to n	1 to n	1 to n	0 to n
Relationship Composition	f, f of f	Exact type sequence	Path of same type	Exact type sequence	Path pattern of different types
Attribute-aware Access Control					
$Common\text{-}friends_k$	√				√
User Attributes		(partial)			√
Relationship Attributes			(partial)		√

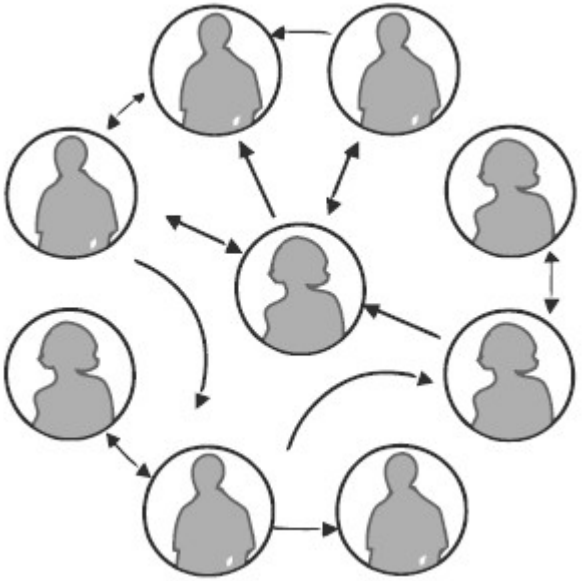
- Passive form of action allows outgoing and incoming action policy
- Path pattern of different relationship types makes policy specification more expressive
- Attribute-aware access control based on attributes of users and relationships

Motivation

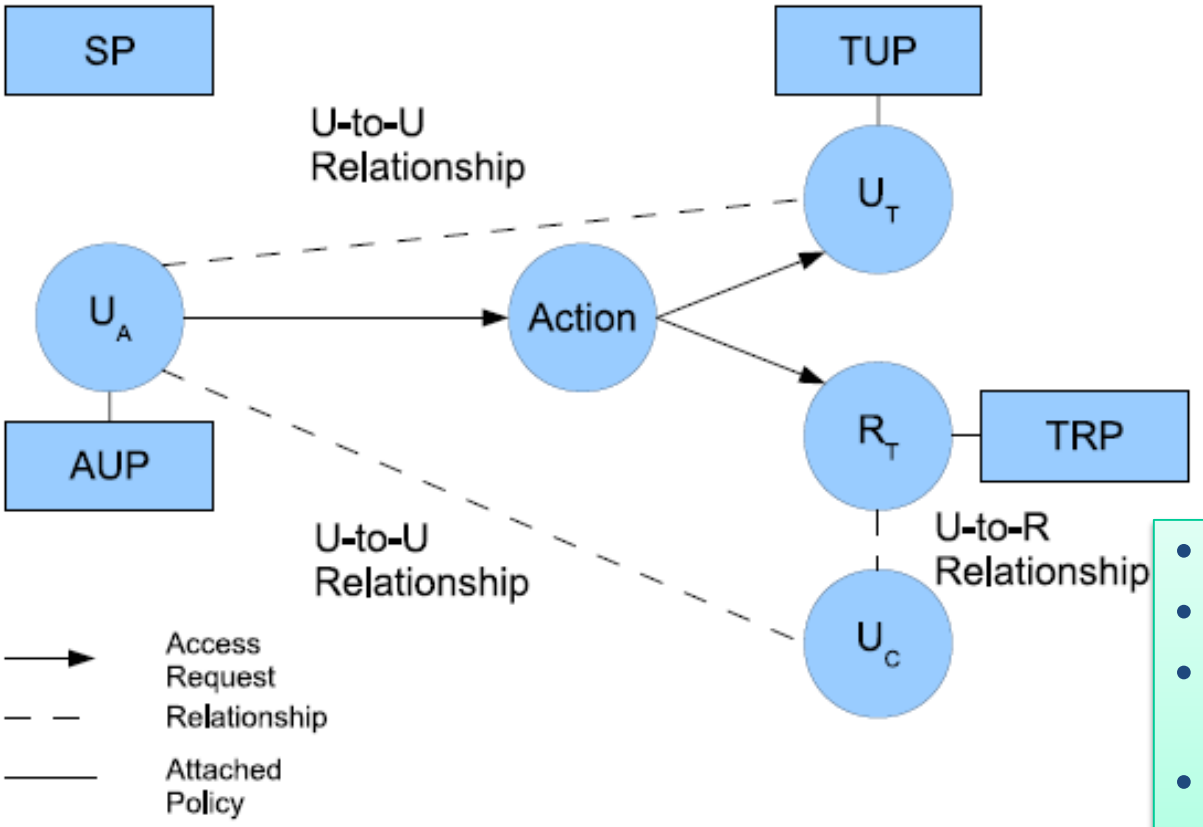
- ReBAC usually relies on type, depth, or strength of relationships, but cannot express more complicated topological information
- ReBAC lacks support for attributes of users, resources, and relationships
- Useful examples include common friends, duration of friendship, minimum age, etc.

$UURAC_A$ Model

- Extended from the UURAC model (DBSec 12)
- Social graph is modeled as a directed labeled simple graph $G=<U, E, \Sigma>$
  - Nodes U as users
  - Edges E as relationships
  - $\Sigma=\{\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_n^{-1}\}$ as relationship types supported



U2U Relationship-based Access Control (UURAC) Model



U_A: Accessing User
U_T: Target User
U_C: Controlling User
R_T: Target Resource
AUP: Accessing User Policy
TUP: Target User Policy
TRP: Target Resource Policy
SP: System Policy

- Policy Individualization
- User and Resource as a Target
- Separation of user policies for incoming and outgoing actions
- Regular Expression based path pattern w/ max hopcounts (e.g., $<u_a, (f*c,3)>$)

Access Request and Evaluation

- Access Request <u_a, *action*, *target*>
  - u_a tries to perform *action* on *target*
  - Target can be either user u_t or resource r_t
- Policies and Relationships used for Access Evaluation
  - When u_a requests to access a user u_t
    - u_a's AUP, u_t's TUP, SP
    - U2U relationships between u_a and u_t
  - When u_a requests to access a resource r_t
    - u_a's AUP, r_t's TRP, SP
    - U2U relationships between u_a and u_c

Policy Representation

Accessing User Policy	$< action, (start, path\ rule)>$
Target User Policy	$< action^{-1}, (start, path\ rule)>$
Target Resource Policy	$< action^{-1}, u_c, (start, path\ rule)>$
System Policy for User	$< action, (start, path\ rule)>$
System Policy for Resource	$< action, (r.typename, r.typevalue), (start, path\ rule)>$

- *action*$^{-1}$ in TUP and TRP is the passive form since it applies to the recipient of action
- TRP has an extra parameter u_c to specify the controlling user
  - U2U relationships between u_a and u_c
- SP does not differentiate the active and passive forms
- SP for resource needs *r.typename, r.typevalue* to refine the scope of the resource

Example

- Alice's policy P_{Alice}:
  - $< poke, \left(u_a, (f\ *, 3)\right) >, < poke^{-1}, \left(u_t, (f, 1)\right) >,$
  - $< read, \left(u_a, (\Sigma\ *, 5)\right) >$
- Harry's policy P_{Harry}:
  - $< poke, \left(u_a, (cf\ *, 5) \vee (f\ *, 5)\right) >, < poke^{-1}, \left(u_t, (f\ *, 2)\right) >$
- Policy of file2 P_{file2}:
  - $< read^{-1}, Harry, (uc, \neg(p+, 2) >$
- System's policy P_{Sys}:
  - $< poke, \left(u_a, (\Sigma\ *, 5)\right) >$
  - $< read, (filetype, photo), \left(u_a, (\Sigma\ *, 5)\right) >$

Attributes in OSNs

- Node attributes
  - Define user's identity and characteristics: e.g., name, age, gender, etc.
- Edge attributes
  - Describe the characteristics of the relationship: e.g., weight, type, duration, etc.
- Count attributes
  - Depict the occurrence requirements for the attribute-based path specification, specifying the lower bound of the occurrence of such path

Attribute-based Policy

- $<quantifier, f(ATTR(N), ATTR(E)), count \geq i>$

∀[+1, -2], age(u) > 18
∃[+1, -1], weight(e) > 0.5
∃{+1, +2, -1}, gender = “male”

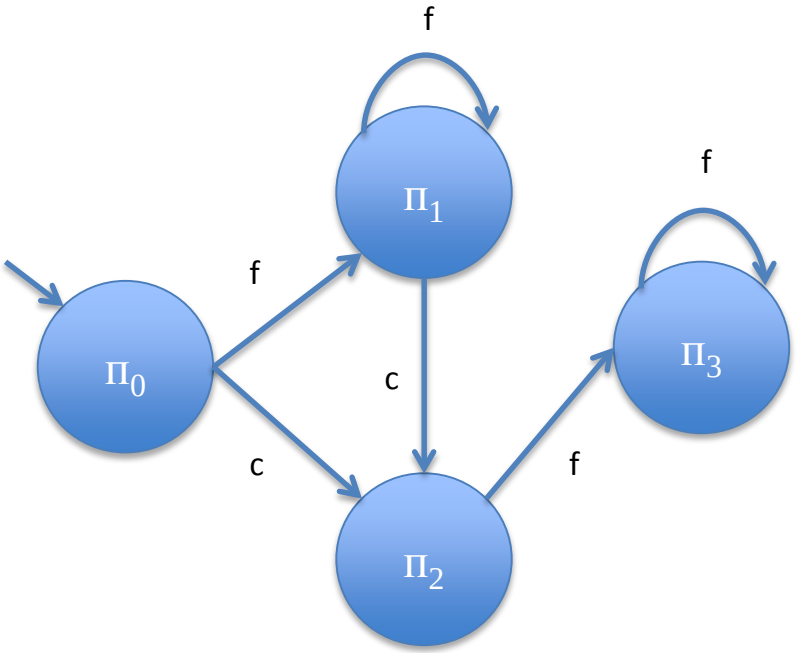
Path-checking Algorithm

- Strategy: DFS
- Parameters: G, path, hopcount, s, t

Access Request: $(\text{Alice}, \text{read}, r_t)$

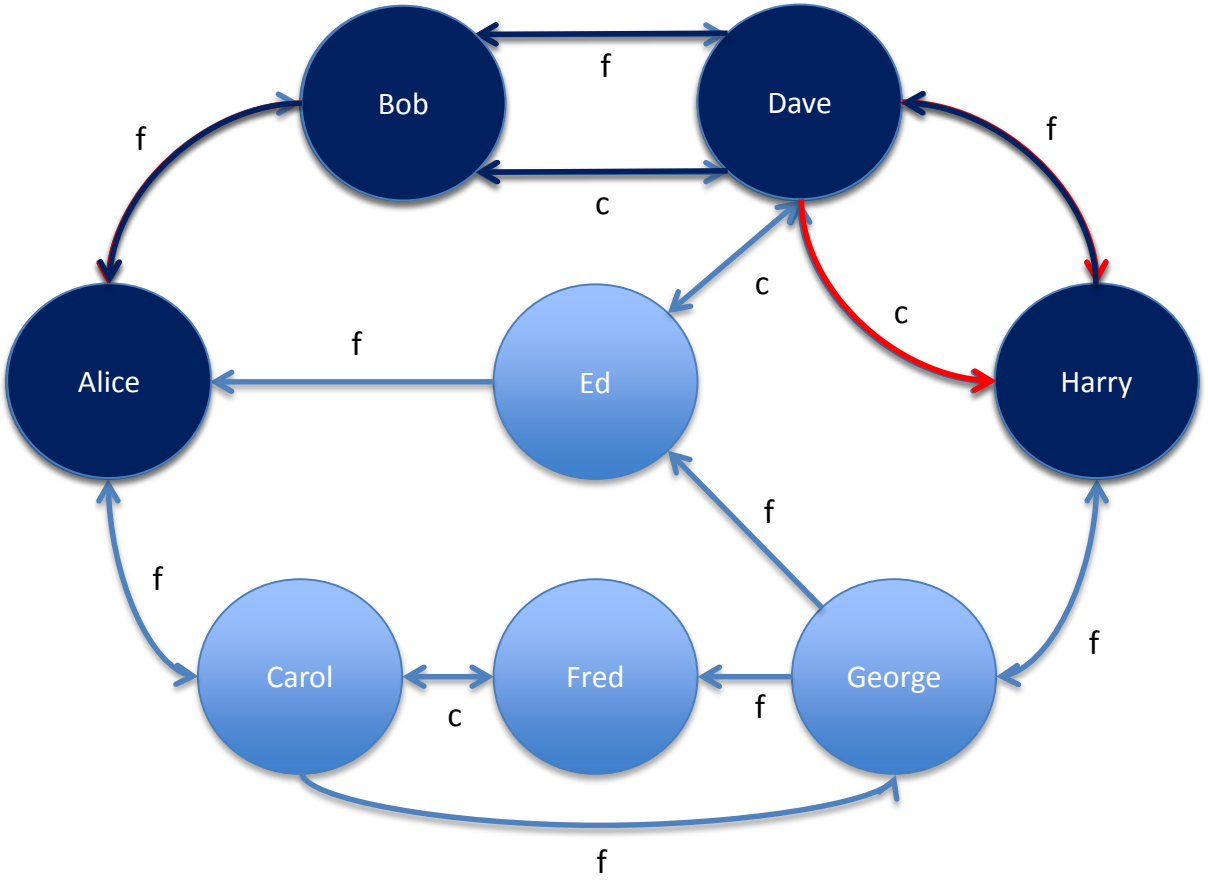
Policy: $(\text{read}^{-1}, r_t, (f*cf*, 3))$

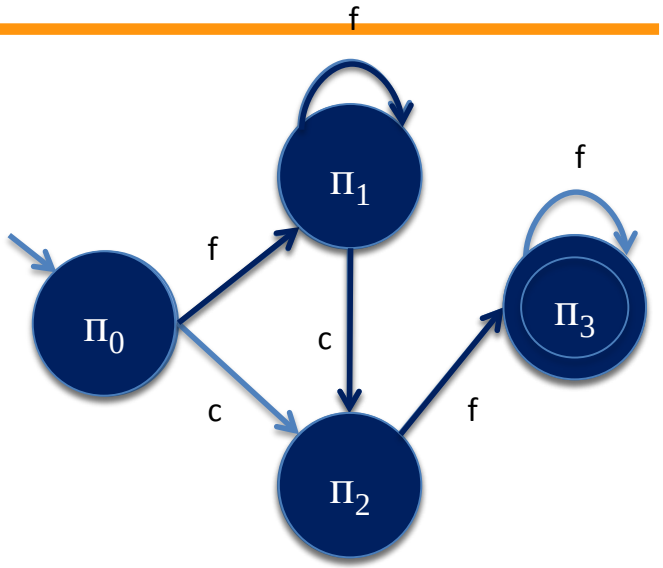
Path pattern: f*cf*
Hopcount: 3



DFA for f*cf*

Path pattern: f*cf*
Hopcount: 3





Case 1: found a path matching pattern, ended at an accepting state

Case 2: found a path matching pattern, ended at an accepting state

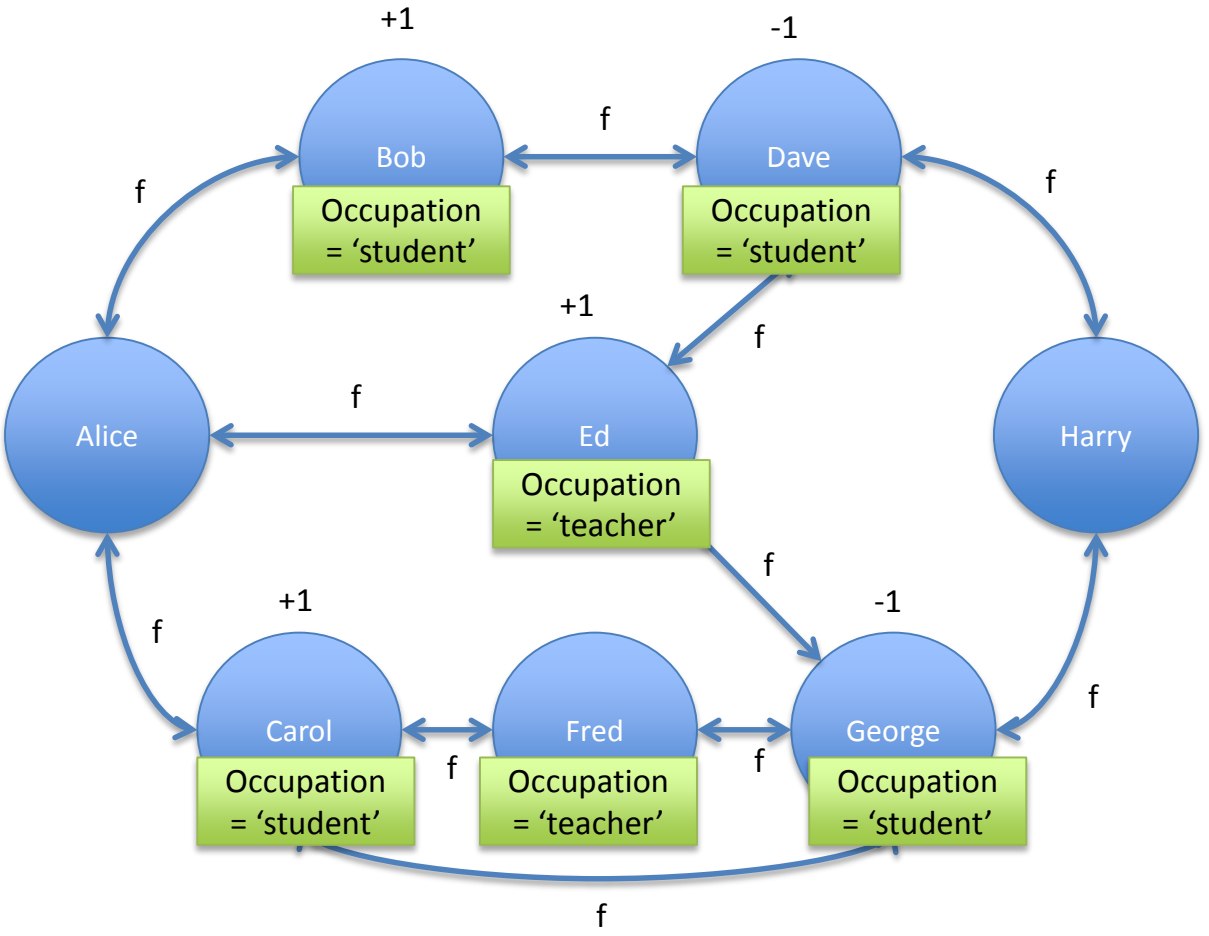
Case 3: found a path matching pattern, did not end at an accepting state

d: 0 1 2 3

currentPath: (H,D,f) (D,B,c) (B,A,f)

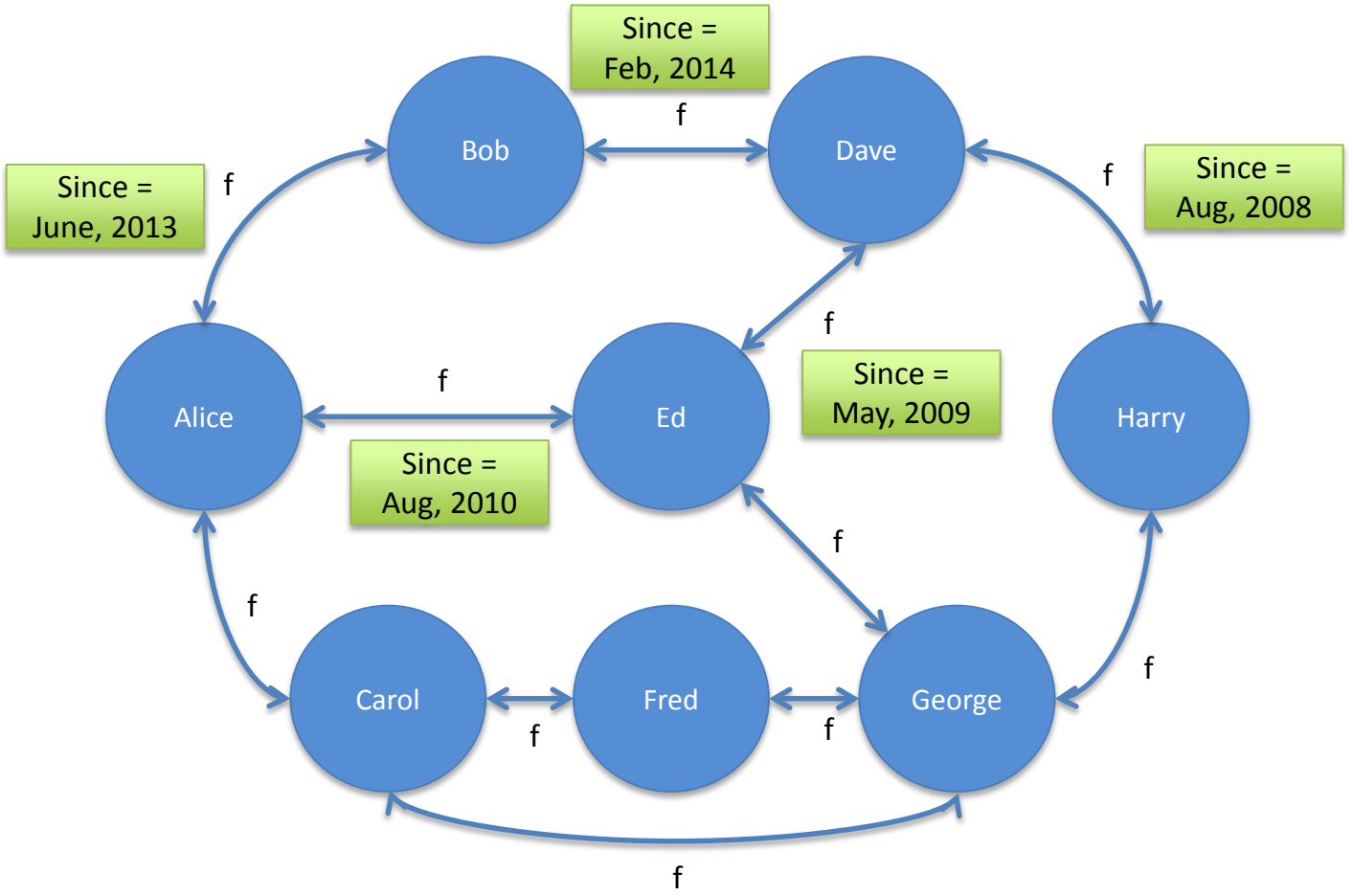
stateHistory: 0 1 2 3

Example: Node Attributes



<access, (u_a, ((f*, 4): ∃[+1, -1], occupation = 'student', count ≥ 3)))>

Example: Edge Attributes



<read, Photo1, (u_a, ((f*, 3): ∀[+1, -1], duration ≥ 3 month, _)))>

Complexity

- Time complexity is bounded between $[O(dmin^{Hopcount}), O(dmax^{Hopcount})]$, where *dmax* and *dmin* are maximum and minimum out-degree of node
  - Users in OSNs usually connect with a small group of users directly, the social graph is very sparse
  - Given the constraints on the relationship types and hopcount limit, the size of the graph to be explored can be dramatically reduced
  - Attribute-based check introduces overhead costs when it finds a possible qualified path, which are proportional to the amount of attributes as well as the type of attribute functions considered

Conclusion

- Presented an extended UURAC model for OSNs
- Formalized the attribute-based policies and the grammar for policy specifications
- Enhanced the path checking algorithm with attribute-awareness

I·C·S
The Institute for Cyber Security

Questions

UTSA