A User-to-User Relationship-based Access Control Model for Online Social Networks

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

Relationship-based Access Control

- Users in OSNs are connected with 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

Problem

- OSNs keep massive resources and support enormous activities for users
- Users want to regulate access to their resources and activities related to them (as a requester or target)
- Some related users also expect control on how the resource or user can be exposed

Motivating Example

- What current FofF approach cannot do?
  - User who is tagged in a photo wants to keep her image private (Related User's Control)
  - Mom doesn't want her kid to become friend with her colleagues (Parental Control)
  - Employee promotes his resume to headhunters without letting his current employer know (Allowing farther users but keeping closer users away)

Characteristics of AC in OSNs

- Policy Individualization
  - Users define their own privacy and activity preferences
  - Related users can configure policies too
  - Collectively used by the system for control decision
- User and Resource as a Target
  - e.g., poke, messaging, friendship invitation, etc.
- User Policies for Outgoing and Incoming Actions
  - User can be either requester or target of activity
  - Allows control on 1) activities w/o knowing a particular resource and 2) activities against the user w/o knowing a particular access requestor
  - e.g., block notification of friend's activities; restrict from viewing violent contents
- Relationship-based Access Control

Solution Approach

- Using regular expression-based path pattern for arbitrary combination of relationship types
- Given relationship path pattern and hopcount limit, graph traversal algorithm checks the social graph to determine access

Related Works

	Fong [7]	Fong [8, 9]	Carminati [6]	Carminati [2, 3]	UURAC
Relationship Category					
Multiple Relationship Types		✓	✓	✓	✓
Directional Relationship		✓	✓		✓
U2U Relationship	✓	✓	✓	✓	✓
U2R 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

- The advantages of this approach:
  - Passive form of action allows outgoing and incoming action policy
  - Path pattern of different relationship types make policy specification more expressive

Contributions

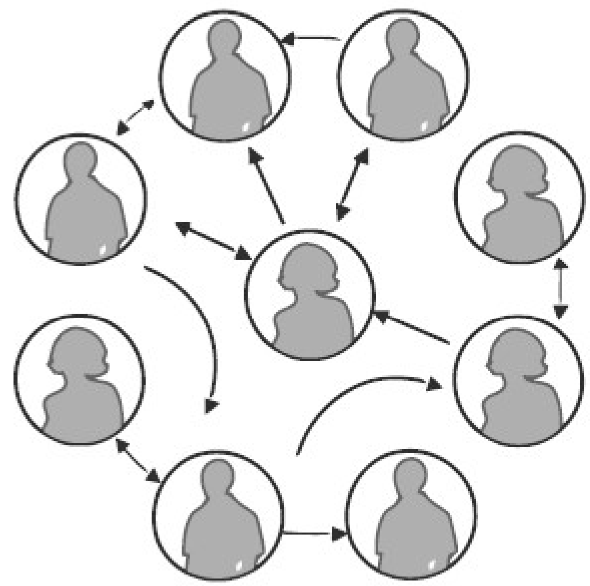
- Provide an access control policy model and access evaluation algorithm for OSNs based on *user-to-user* relationships with
  - Greater generality and flexibility of policy specification
  - Effective evaluation of policy predicate

Outline

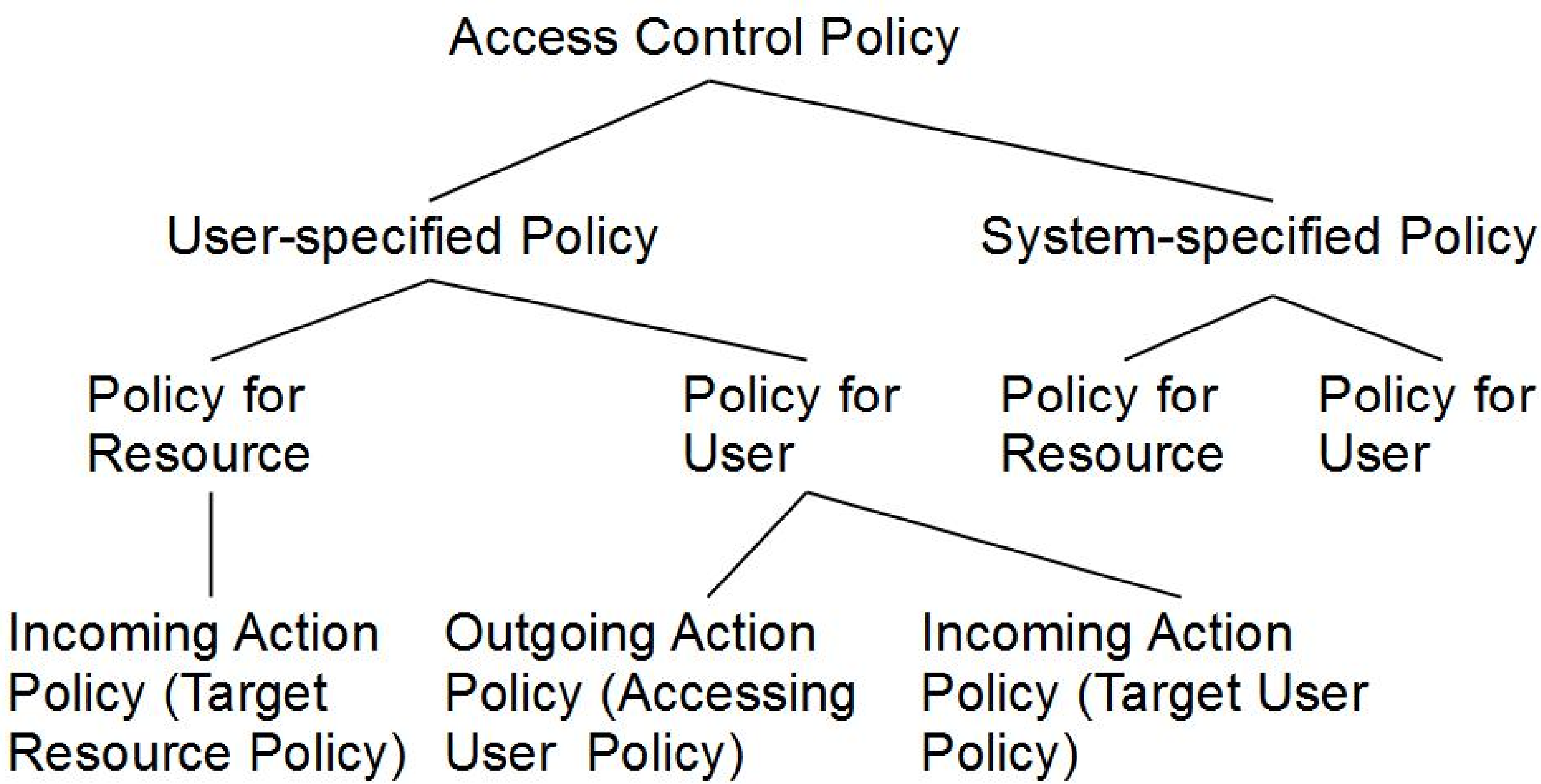
- Motivation
- UURAC Model Foundation
- UURAC Policy Specification
- Path-checking Algorithm
- Conclusions

Social Networks

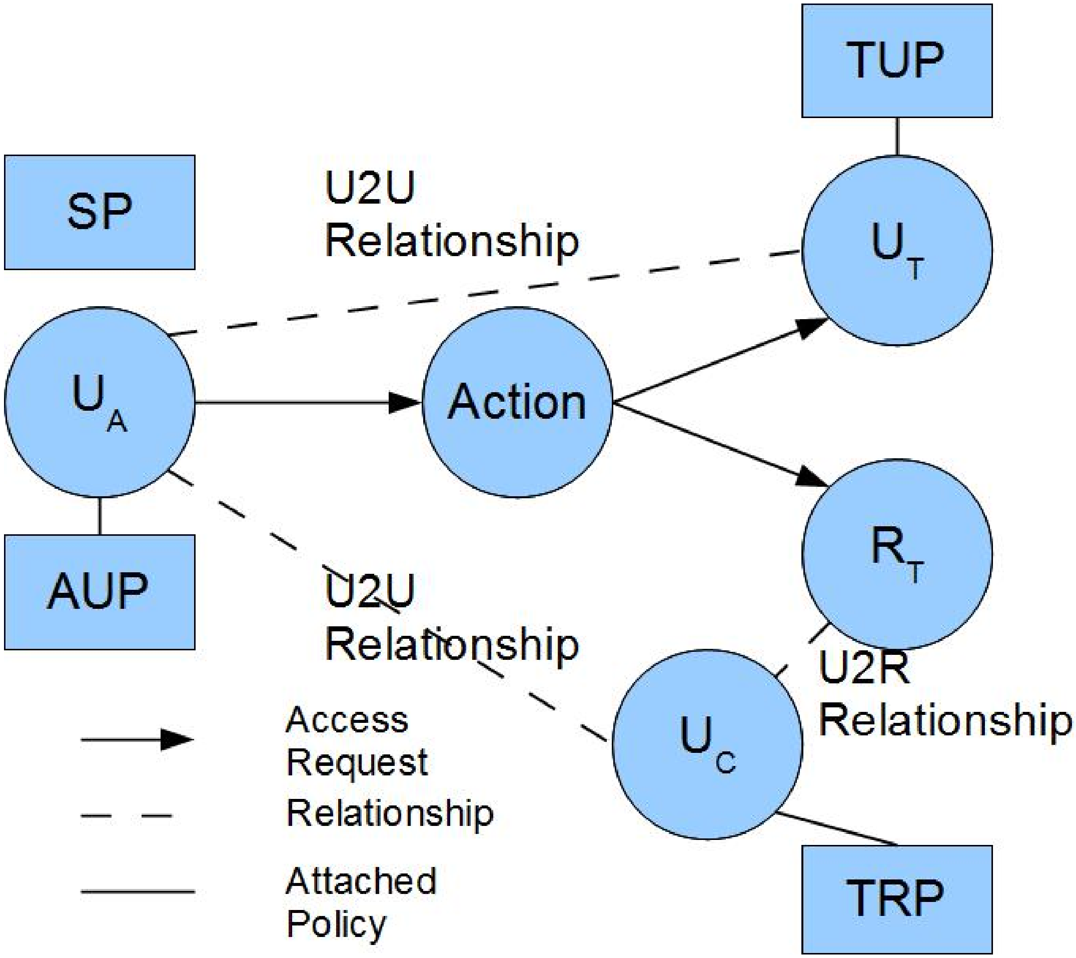
- Social graph is modeled as a directed labeled simple graph *G*=<*U*, *E*, *Σ*>
  - – Nodes *U* as users
  - – Edges *E* as relationships
  - – $\Sigma=\{\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_n^{-1}\}$ as relationship types supported



Policy Taxonomy



UURAC Model Components



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

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 (associated with u_c), SP
    - U2U relationships between u_a and u_c

Outline

- Motivation
- UURAC Model Foundation
- UURAC Policy Specification
- Path-checking Algorithm
- Conclusions

Policy Representations

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

- *$action^{-1}$* in TUP and TRP is the passive form since it applies to the recipient of action
- TRP has an extra parameter r_t to distinguish the actual target resource it applies to
  - owner(r_t)→ a list of u_c→U2U relationships between u_a and u_c
- SP does not differentiate the active and passive forms
- SP for resource needs *r.type* to refine the scope of the resource

Graph Rule Grammar

$GraphRule ::= \text{“(”} < StartingNode > \text{“,”} < PathRule > \text{“)”}$
$PathRule ::= < PathSpecExp > | < PathSpecExp >< Connective >< PathRule >$
$Connective ::= \vee | \wedge$
$PathSpecExp ::= < PathSpec > | \neg < PathSpec >$
$PathSpec ::= \text{“(”} < Path > \text{“,”} < HopCount > \text{“)”} | \text{“(”} < EmptySet > \text{“,”} < Hopcount > \text{“)”}$
$HopCount ::= < Number >$
$Path ::= < TypeExp > | < TypeExp >< Path >$
$EmptySet ::= \emptyset$
$TypeExp ::= < TypeSpecifier > | < TypeSpecifier >< Wildcard >$
$StartingNode ::= u_a | u_t | u_c$
$TypeSpecifier ::= \sigma_1 | \sigma_2 | .. | \sigma_n | \sigma_1^{-1} | \sigma_2^{-1} | .. | \sigma_n^{-1} | \Sigma \text{ where } \Sigma = \{\sigma_1, \sigma_2, .., \sigma_n, \sigma_1^{-1}, \sigma_2^{-1}, .., \sigma_n^{-1}\}$
$Wildcard ::= \text{“}*\text{”} | \text{“?”} | \text{“}+\text{”}$
$Number ::= [0-9]+$

Example

- Alice's policy P_{Alice}: $< poke, (u_a, (f*, 3))> < poke^{-1}, (u_t, (f, 1))> < read, (u_a, (\Sigma*, 5))> < read^{-1}, file1, (u_c, (cf*, 4))>$
- Harry's policy P_{Harry}: $< poke, (u_a, (cf*, 5) \vee (f*, 5))> < poke^{-1}, (u_t, (f*, 2))> < read^{-1}, file2, (u_c, \neg(p+, 2)>$
- System's policy P_{Sys}: $< poke, (u_a, (\Sigma*, 5))> < read, photo, (u_a, (\Sigma*, 5))>$

- "Only Me"
  - <poke, (ua, (Ø, 0))> says that ua can only poke herself
  - <poke^{-1}, (ut, (Ø, 0))> specifies that ut can only be poked by herself
- The Use of Negation Notation
  - (fffc ^ ¬fc) allows the coworkers of the user's distant friends to see, while keeping away the coworkers of the user's direct friends

Policy Collecting

- To authorize (u_a, action, target) if target = u_t
  - E.g., (Alice, poke, Harry)

AUP

TUP

SP

P_{Alice}

<poke, (ua, (f*,3))>

< poke^{-1}, (ua, (f*,3))>

P_{Harry}

<poke, (ua, (cf*,5)$\vee$(f*,5))>

<poke^{-1}, (ut, (f*,2))>

P_{Sys}

<poke, (ua, (Σ*,5))>

Policy Collecting

- To authorize (u_a, action, target) if target = r_t
  - Determine the controlling user for r_t:
    - $u_c \leftarrow owner(r_t)$
  - E.g., (Alice, read, file2)

AUP

P_{Alice}

<read, (u_a, (Σ*, 5))>

<read^{-1}, file1, (u_c, (cf*, 4))>

TRP

P_{Harry}

<read^{-1}, file2, (u_c, ¬(p+, 2))>

SP

P_{Sys}

<read, photo, (ua, (Σ*, 5))>

Policy Extraction

It determines the starting node, where the evaluation starts

The other user involved in access becomes the evaluating node

- Policy: <*act* , *graph rule*>
- Graph Rule: *start*, *path rule*

↓

- Path Rule: *path spec* ∧|∨ *path spec*

↓

- Path Spec: *path*, *hopcount*

Path-check each path spec using Algorithm 2 (introduced in detail later)

Policy Evaluation

- Evaluate a combined result based on conjunctive or disjunctive connectives between path specs
- Make a collective result for multiple policies in each policy set.
  - Policy conflicts may arise. We assume system level conflict resolution strategy is available (e.g., disjunctive, conjunctive, prioritized).
- Compose the final result from the result of each policy set (AUP, TUP/TRP, SP)

Outline

- Motivation
- UURAC Model Foundation
- UURAC Policy Specification
- Path-checking Algorithm
- Conclusions

Brief Intro

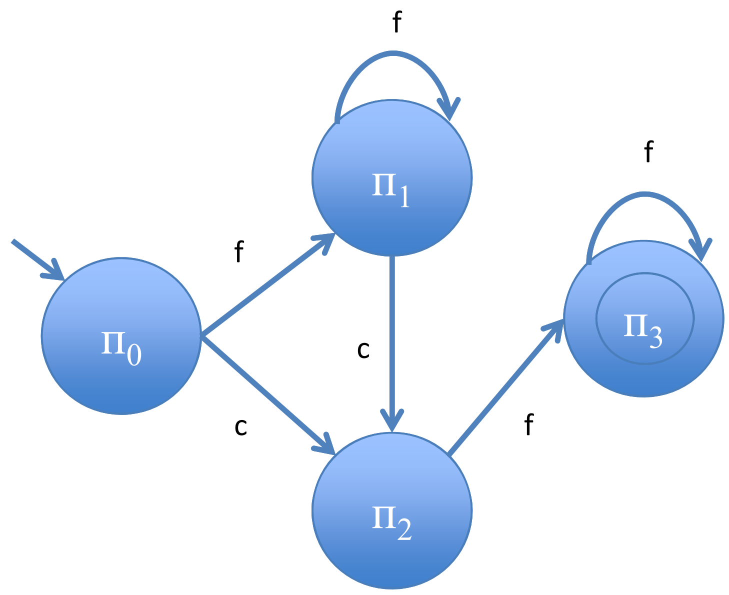
- Parameters: G, path, hopcount, s, t
- Traversal Order: Depth-First Search
  - Why not BFS?
    - Activities in OSN typically occur among people with close distance
    - DFS needs only one pair of variables to keep the current status and history of exploration
    - Hopcount limit prevents DFS from lengthy useless search

Initiation

Access Request: (Alice, read, r_t)

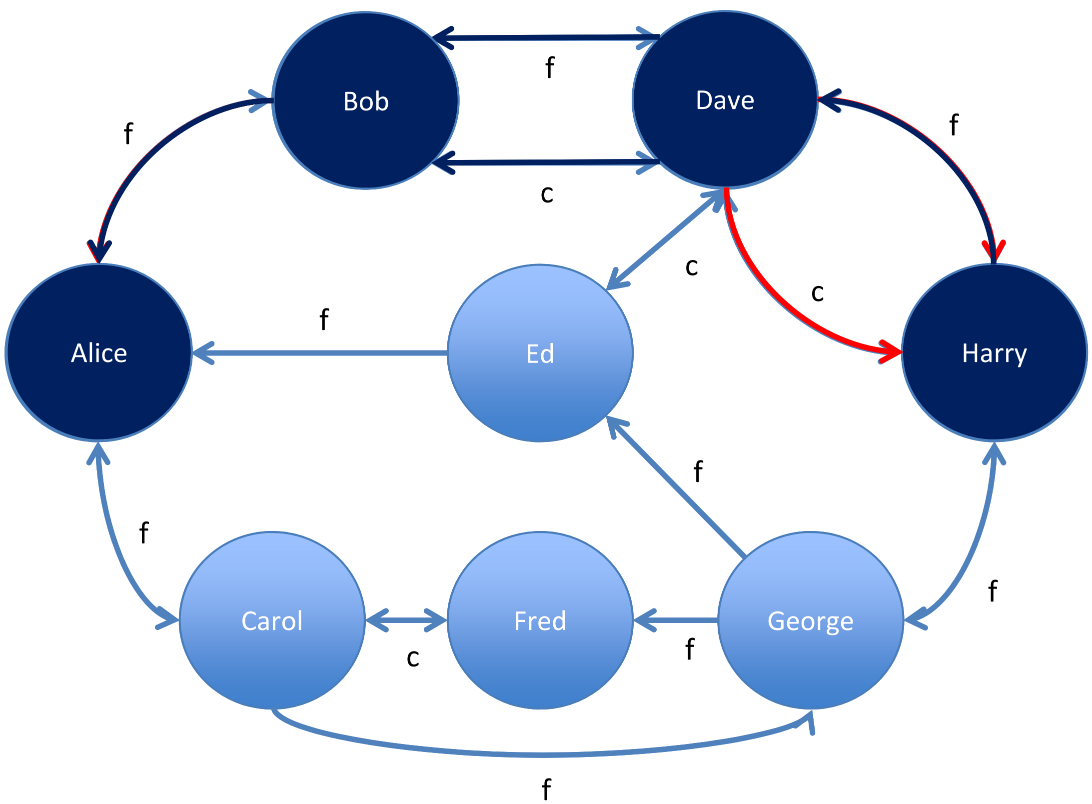
Policy: ($read^{-1}$, r_t, (f*cf*, 3))

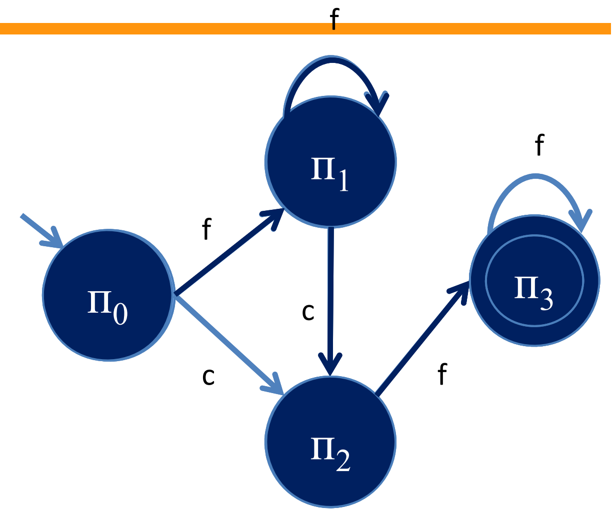
Path pattern: f*cf*
Hopcount: 3



DFA for f*cf*

Path pattern: f*cf*
Hopcount: 3





Case 3: [illegible] matching [illegible] an [illegible] DFA not at an accepting state

d: 3

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

stateHistory: 0123

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

Outline

- Motivation
- UURAC Model Foundation
- UURAC Policy Specification
- Path-checking Algorithm
- Conclusions

Summary

- Proposed a U2U relationship-based model and a regular expression-based policy specification language for OSNs
- Provided a DFS-based path checking algorithm

Future Work

- Possible extensions:
  – Exploit U2R and R2R relationships
  – Incorporate predicate expressions for attribute-based control
  – Capture unconventional relationships

Questions