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

World-Leading Research with Real-World Impact!
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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Relationship Types</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Directional Relationship</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>U2U Relationship</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>U2R Relationship</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Model Characteristics</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Policy Individualization</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>User &amp; Resource as a Target</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Outgoing/Incoming Action Policy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Relationship Composition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Relationship Depth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Relationship Composition</td>
<td>0 to 2 f, f of f</td>
<td>0 to n exact type sequence</td>
<td>1 to n path of same type</td>
<td>1 to n exact type sequence</td>
<td>0 to n path pattern of different types</td>
</tr>
</tbody>
</table>

- 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 $G=\langle U, E, \Sigma \rangle$
  - Nodes $U$ as users
  - Edges $E$ as relationships
  - $\Sigma=\{\sigma_1, \sigma_2, ..., \sigma_n, \sigma_1^{-1}, \sigma_2^{-1}, ..., \sigma_n^{-1}\}$ as relationship types supported
UURAC Model Components

UA: Accessing User
UT: Target User
UC: Controlling User
RT: Target Resource
AUP: Accessing User Policy
TUP: Target User Policy
TRP: Target Resource Policy
SP: System Policy

World-Leading Research with Real-World Impact!
Access Request and Evaluation

• Access Request \(<u_a, \text{action}, \text{target}>\)
  – \(u_a\) tries to perform \text{action} on \text{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

<table>
<thead>
<tr>
<th>Policy Representation</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing User Policy</td>
<td>( \langle \text{action}, (\text{start}, \text{path rule}) \rangle )</td>
</tr>
<tr>
<td>Target User Policy</td>
<td>( \langle \text{action}^{-1}, (\text{start}, \text{path rule}) \rangle )</td>
</tr>
<tr>
<td>Target Resource Policy</td>
<td>( \langle \text{action}^{-1}, \text{r}_t, (\text{start}, \text{path rule}) \rangle )</td>
</tr>
<tr>
<td>System Policy for User</td>
<td>( \langle \text{action}, (\text{start}, \text{path rule}) \rangle )</td>
</tr>
<tr>
<td>System Policy for Resource</td>
<td>( \langle \text{action}, \text{r.type}, (\text{start}, \text{path rule}) \rangle )</td>
</tr>
</tbody>
</table>

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

GraphRule ::= “(” <StartingNode> “,” <PathRule> “)”
PathRule ::= <PathSpecExp> | <PathSpecExp><Connective><PathRule>
Connective ::= ∨|∧
PathSpecExp ::= <PathSpec> |¬<PathSpec>
PathSpec ::= “(” <Path> “,” <HopCount> “)”|“(” <EmptySet> “,” <Hopcount> “)”
HopCount ::= <Number>
Path ::= <TypeExp> | <TypeExp><Path>
EmptySet ::= ∅
TypeExp ::= <TypeSpecifier> | <TypeSpecifier><Wildcard>
StartingNode ::= u_a|u_t|u_c
TypeSpecifier ::= σ_1|σ_2|..|σ_n|σ_1^{-1}|σ_2^{-1}|..|σ_n^{-1}|Σ where Σ = {σ_1, σ_2,.., σ_n, σ_1^{-1}, σ_2^{-1},.., σ_n^{-1}}
Wildcard ::= “*”|“?”|“+”
Number ::= [0 – 9]+
Example

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

- “Only Me”
  • $<\text{poke}, (ua, (\emptyset, 0))>$ says that $ua$ can only poke herself
  • $<\text{poke}^{-1}, (ut, (\emptyset, 0))>$ specifies that $ut$ can only be poked by herself
- The Use of Negation Notation
  • $(\text{fff}c \land \neg fc)$ 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 \((u_a, \text{action}, \text{target})\) if \(\text{target} = u_t\)
  
  - E.g., (Alice, poke, Harry)

```
\text{P}_{\text{Alice}} <\text{poke}, (u_a, (f^*,3))> \\
<\text{poke}^{-1}, (u_a, (f^*,3))>
```

```
\text{P}_{\text{Harry}} <\text{poke}, (u_a, (cf^*,5) \vee (f^*,5))> \\
<\text{poke}^{-1}, (u_t, (f^*,2))>
```

```
\text{P}_{\text{Sys}} <\text{poke}, (u_a, (\Sigma^*,5))>
```
Policy Collecting

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

- AUP
- TRP
- SP

---

World-Leading Research with Real-World Impact!
Policy Extraction

- Policy: \(<\text{act, type, graph rule}>\)
- Graph Rule: \(\text{start, path rule}\)
- Path Rule: \(\text{path spec } \land | \lor \text{ path spec}\)
- Path Spec: \(\text{path, hopcount}\)

The other user involved in access becomes the evaluating node.

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

World-Leading Research with Real-World Impact!
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, rt)

Policy: (read⁻¹, rt, (f*cf*, 3))

Path pattern: f*cf*
Hopcount: 3
Path pattern: f*cf*
Hopcount: 3

Case 1: found a matching path up to the point of the next character of the pattern, but the DFA not at an accepting state

Case 2: found a matching path and DFA reached an accepting state

Case 3: currentPath matches the prefix of the pattern, but DFA not at an accepting state

d: Ø
currentPath: (H,D,f)(D,B,c)(B,A,f)
stateHistory: 0123
Complexity

- Time complexity is bounded between $[O(d_{min}^{Hopcount}), O(d_{max}^{Hopcount})]$, where $d_{max}$ and $d_{min}$ 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