PRIVACY @INSTAGRAM

Carrie Jiang — 2018
PRIVACY @ INSTAGRAM

Agenda

1 The initiative

2 The framework

3 Privacy policy inference
INSTAGRAM USER GROWTH


80 M      |  300 M |  500 M |  800 M |  800 M
#hashtags
def filter_activities(activities, viewer):
    result = []
    for act in activities:
        if is_blocking(viewer, act.actioner) or blocked(viewer, act.actioner):
            continue
        if act.type == 'LIKE':
            if is_blocking(viewer, act.target_user) or \
            blocked(viewer, act.target_user):
                continue
            if act.target_user.is_private and \
            not is_following(viewer, act.target_user):
                continue
            if not can_view_media(viewer, act.media):
                continue
            result.append(act)
    return result
CENTRALIZED PRIVACY SERVICE

- Endpoint level privacy check
- Data level privacy check
ENDPOINT LEVEL PRIVACY

User Timeline

{data, privacy_config} → {filtered data} → Privacy Service → {filtered data} → {data, privacy_config}

Explore
DATA LEVEL PRIVACY

Notification

User Timeline

Explore

Privacy Check

Data Storage
## COMPARISON

<table>
<thead>
<tr>
<th></th>
<th>Correctness</th>
<th>Performance</th>
<th>Developer Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Endpoint Level</strong></td>
<td>Flexible per use case</td>
<td>Overfetch</td>
<td>Master privacy settings</td>
</tr>
<tr>
<td></td>
<td>High maintenance</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Level</strong></td>
<td>Enforced everywhere</td>
<td>Proper caching</td>
<td>Implicit and automatic</td>
</tr>
<tr>
<td></td>
<td>Low maintenance</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DATA LEVEL PRIVACY

- Data Fetching API
- Privacy Check
- Data Storage

- Data Fetching API
- Privacy Check
- Data Storage
- Data Storage
- Data Storage
TERMINOLOGY

- Viewer Context

- Privacy Rule
  - allow or deny decision on privacy scenario

- Privacy Policy
  - allow or deny decision on an object based on a series of privacy scenarios
class DenyIfUserIsBlockedRule:
    DENY_RULING = Ruling.deny(
        'DenyIfUserIsBlockedRule',
        'Deny if target user and viewer are blocking each other'
    )

    def evaluate(
        self,
        viewer_context: ViewerContext,
        node: "NodeUser",
    ) -> Ruling:
        if node.blocked(viewer_context.user) or node.is_blocking(viewer_context.user):
            return DENY_RULING
        return skip_ruling
privacy policy on user

read_policy:
- deny_if_user_is_blocked_rule
- deny_if_user_is_gated_rule
- allow_if_public_user_rule
- always_deny_rule
DELEGATION

read_policy:
- deny_if_cannot_see_media_owner_rule
- allow_if_media_owner_is_public_rule
- always_deny_rule
EDGE PRIVACY

User  Favorite Media  Media
FIELD PRIVACY

User

Username
Follower Count
Basic Profile

Public Fields

Phone
Email
Follower

Private Fields
DEPLOYMENT

- Focus on core types that responsible for most of the data access
- Always stricter or equal to existing privacy checks
- REALLY Look out for performance
POLICY INFRINGEMENT

- Sample data accesses per endpoint
- Build relationship graph between viewer and data
- Analyze common relationship
## POLICY INFERENCEx

<table>
<thead>
<tr>
<th>Device ID</th>
<th>Viewer</th>
<th>Owner</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Joe</td>
<td>Joe</td>
<td>user.settings</td>
</tr>
<tr>
<td>2</td>
<td>Alex</td>
<td>Alex</td>
<td>explore</td>
</tr>
<tr>
<td>3</td>
<td>Mike</td>
<td>Mike</td>
<td>user.login</td>
</tr>
<tr>
<td>4</td>
<td>Emily</td>
<td>Emily</td>
<td>user.settings</td>
</tr>
<tr>
<td>5</td>
<td>None</td>
<td>Kevin</td>
<td>user.foo</td>
</tr>
</tbody>
</table>
CONCLUSION

- Data level privacy check
  - correctness
  - develop velocity

- Privacy policy inference
  - infer policy
  - detection