DelF: Safeguarding deletion correctness in Online Social Networks

Emmie He, Singh Saluja, Eleanor Eng
Context

- Why is deletion important?
- For OSNs, what are some challenges with deletion?
Vulnerability Incidents (6.3)

- October 2019: developers changed how photos are handled
- The edge (deep) was associated with the thread (photo)
- Instagram bug - 17,000 photos deleted and millions of likes and comments.
  - Restoration ran for 10 days
  - Multiple engineering teams involved
- Exploitation was possible because DELF privilege escalation checks were not enforced in Instagram when the bug occurred
Problem

- Deletion is an important factor of OSNs
- Complications with deletion result in:
  - Mistakenly retaining data
  - Removing the wrong data
  - Security vulnerabilities
- No existing omission-catching mechanism
TAO – Facebook’s Graph Database

The Graph Your Network application inserts your Twitter activity into Neo4j.

Here is a data model of the graph:

```latex
// Graph of some of your mentions
MATCH
(u:Me:User)-[p:POSTS]->(t:Tweet)-[:Mentions]->(m:User)
WITH
t, u, p, t, m,
COUNT(m.screen_name) AS count
ORDER BY
count DESC
RETURN
u, p, t, m
LIMIT 10
```
DelF Design

- An Overview
- (Let’s discuss!)
Key Terms

- **Subgraph deletions**: application-level
- **Point deletes**: row or object-level
- **Key-value model**: data indexed by arbitrary strings
- **Graph model**: data forms a graph
- **Dangling data**: reference to deleted object and its parent object
- Provides annotations for deletion control
  - **Shallow**: deletes source object
  - **Deep**: follows edge to delete associated object
  - **Refcount**: cascades when last edge to target is deleted
Test Case: Facebook

- **Infrastructure**
  - Data stores: TAO, Everstore, MySQL, ZippyDB
  - Developers are required to define their data types

- **Deletion policy**
  - Deletions take a max of 90 days
  - Deleted data persists for up to 90 days to allow for recovery

- **Findings**
  - Deletion was proactively handled for only 20% of the edge types created
  - Deletion is rarely a requirement when prototyping -> often only thought of retroactively
  - Developers misclassified edge types with an overall precision of 97.6%
  - Developers incorrectly annotate about 2 edge types per day
  - Mistakes that result in inadvertent deletion are about twice as common than those that result in inadvertent data retention
Key Contributions

- Case study quantifying rate of mistakes of developer actions at Facebook
- Development of and justification for DelF
- Demonstration of DelF’s capabilities
- Deployment of DelF at Facebook

<table>
<thead>
<tr>
<th>Annotation</th>
<th>True Pos.</th>
<th>False Pos.</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>shallow</td>
<td>239</td>
<td>5</td>
<td>98.0%</td>
</tr>
<tr>
<td>deep</td>
<td>87</td>
<td>3</td>
<td>96.7%</td>
</tr>
<tr>
<td>refcount</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Overall</strong></td>
<td><strong>326</strong></td>
<td><strong>8</strong></td>
<td><strong>97.6%</strong></td>
</tr>
</tbody>
</table>

- The first study to measure developer actions such as obligation to delete data, correct deletion, etc.
Evaluation

- The first framework of its kind
- At Facebook, found thousands of omissions and mistakes in deletion
Advocate

(Eleanor can do this)

- Framework was tested in real env: Facebook
- Quantifies performance with and without DeLF
- Addresses both deletion and recovery
- Contains interesting findings on previous unexplored topics
Critic