Differentially-Private Control-Flow Node Coverage for Software Usage Analysis

Hailong Zhang, Sufian Latif, Raef Bassily, Atanas Rountev
Problem and motivation

DP control-flow node coverage analysis

Results and conclusions
Software usage analysis in the wild

Software collect numerous usage data from users
- How frequently certain things happen across all users?
- E.g., common behavior analysis, finding popular features, locating performance bottlenecks, ...

Example: mobile app analytics
Data privacy in the wild

Collect individual’s data to calculate statistics
- Data breaches, unethical business practices, rogue employees, powerful adversarial data analysis, ...

How can we do software usage analysis with principled privacy protection?

Solution: Differential privacy
Focus: CFG node coverage analysis

CFG is a graph representation of the control flow during the execution of a program

- Node: a statement, a code block, a function/method, a coarse-grained software component, ...
- Edge: temporal relationship between nodes
Focus: CFG node coverage analysis

The execution at each user covers a subset of nodes and edges in the CFG

- Node coverage analysis collects coverage of nodes from users

Goal: for each graph node, what is the number of users who have executed it
Focus: CFG node coverage analysis

Solution: Node coverage analysis with local differential privacy
Problem and motivation

DP control-flow node coverage analysis

Results and conclusions
Node coverage at each user

Each user $i$ holds a coverage vector $c_i$

- One bit for each CFG node
- The bit is 1 if the corresponding node is covered at run time by the user

$$c_i = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$
The differential privacy protection

**Goal:** Do not leak information about whether a particular node is covered or not during data collection

*Each user runs a local randomizer* $R$

\[
\begin{align*}
c_i & = [1 \quad 1 \quad 1 \quad \ldots] \\
R(c_i) & = R
\end{align*}
\]
The differential privacy protection

For any CFG node $v$, local randomizer $R$ guarantees the *indistinguishability* between any pair of coverage vectors that are *neighbors* w.r.t. $v$

- $R$ ensures that anyone observing the output cannot determine *whether $v$ is covered or not* at user $i$

$\mathbf{c}_i = [1 \ 1 \ 1 \ 1 \ ...]$  

$\mathbf{c}_i' = [1 \ 0 \ 0 \ 1 \ ...]$  

*How to define neighbors?*
Example: Hiding the coverage of \textbf{a}

\begin{itemize}
  \item \textbf{b} can only be invoked when \textbf{a} is executed.
\end{itemize}

\begin{tabular}{cccccc}
  \hline
  \textbf{start} & \textbf{a} & \textbf{b} & \textbf{c} & \ldots \\
  \hline
  \textbf{start} & 1 & 1 & 0 & \ldots \\
  \textbf{c} & 0 & 1 & 0 & \ldots \\
  \textbf{a} & 0 & 0 & 0 & \ldots \\
  \textbf{b} & 0 & 0 & 0 & \ldots \\
  \hline
\end{tabular}
Coverage vector neighbors

This correlation is captured by the **dominator tree** of a CFG

- In a CFG, a node \( d \) **dominates** a node \( m \) if every path from the start node to \( m \) goes through \( d \)

- **Dominator tree**: For any node in the tree, the set of its ancestors is exactly the set of its dominators

Dominator tree of the CFG in the example

The sensitivity of \( a \) in \( c_i \) is 2
Definition of local randomizer

Let $S$ be an upper bound of all sensitivities of all users

- Randomizer $R$ flips each bit in $c_i$ with probability $\frac{1}{1+e^{\epsilon/S}}$

- $S \uparrow$, protection $\uparrow$, accuracy $\downarrow$

$$c_1 = [1011 \ldots] \rightarrow R(c_1)$$
$$c_2 = [1001 \ldots] \rightarrow R(c_2)$$
$$\vdots$$
$$c_n = [1110 \ldots] \rightarrow R(c_n)$$

$$h = \sum_i R(c_i)$$

$$\hat{f} = \frac{(1 + e^{\epsilon/S})h - n}{e^{\epsilon/S} - 1}$$
Selecting the upper bound

1. The baseline approach: \( S = |N| - 1 \)

2. Tighter bound via restricted sensitivity
   - Map vectors to restricted domain with lower sensitivity

3. Relaxed indistinguishability
   - Calibrate the strength of indistinguishability of neighbors
Problem and motivation
DP control-flow node coverage analysis
Results and conclusions
Experimental setup

15 popular Android apps
- Coverage analysis for GUI screen view graphs and call graphs
- Automatically simulate 1000 users per app

Metrics
- Mean error: average error of each node
\[ \text{ME} = \frac{\sum_v |f(v) - \hat{f}(v)|}{|N|} \]
Experimental evaluation

Fundamental trade-off between privacy and accuracy

![Screen graph and call graph comparison chart](chart.png)

- **ME**
  - Screen graph:
    - Baseline: 1.5x
    - Tighter: 10x
    - Relaxed: 0x
  - Call graph:
    - Baseline: 2x
    - Tighter: 19x

Max: 1000
Experimental evaluation

Fundamental trade-off between privacy and accuracy

![Graph showing ME comparisons for screen and call graphs]
Conclusions

Privacy is highly needed for software usage analysis.

Differential privacy is an appealing and applicable tool.

Existing DP techniques cannot be directly applied; domain-specific knowledge is needed, e.g., dominators.

The proposed approaches can achieve high accuracy with meaningful privacy protection.
Differentially-Private Control-Flow Node Coverage for Software Usage Analysis

Hailong Zhang   zhang.4858@osu.edu
Sufian Latif    latif.28@osu.edu
Raef Bassily    bassily.1@osu.edu
Atanas Rountev  rountev.1@osu.edu