# CS5430 Project 1: Access Control Analysis

Shuochi Huang (sh2534), Shunqi Mao (sm2784), Pu Zhao (pz551)

## Key algorithms and data structures

### Data structures

We used a directed graph to store principals and files, where every principal and every file is a vertex, and every privilege is an edge. If a principal S has read access to a file O, there is a directed edge from O to S. And if a principal S has write access to a file O, there is a directed edge from S to O.
The graph is implemented using a hash table, where keys are vertices and values are a set of outbound edges and a set of inbound edges. If a vertex is a file, we omit the inbound edges.

Viable alternatives for high level data structures include access control list and capability list. Using these methods will cause the time complexity of leakage analysis to significantly grow because we have to repeatedly and alternately search for subjects and objects to determine if multiple subjects can conspire and leak the file through multiple reads and writes.

Also, for all of the above data structures, instead of storing privileges as a hashset for each principal and file, it is viable to store it as an array. By doing so, the time complexity will grow from O(1) to O(n) for Query operation.

### Algorithms

Add:
Add appropriate edges to the graph to maintain the data structures. Add vertices if the subject or the object does not exist.

Query:
Find if there is an edge from O to S if priv is R, or if there is an edge from S to O if priv is W.

Leak:
Breadth-First Search algorithm is used to determine if there exists a directed path from F1 to F2. If exists, return YES, otherwise NO.

# Asymptotic complexity

V := # of principals and files
E := # of privileges

## Time complexity

Add: O(1) for R and W, O(V) for G

Query: O(1)

Leak: O(V + E)

## Space complexity

Add: O(1) for R and W, O(V) for G

Query: Does not require additional memory.

Leak: O(V) for queue and set used by BFS

# UGCLab running times

## Executions

Input1:
Add, s1, o1, R
Add, s2, o1, W
Query, s1, o1, R
Leak, o1, o2

Execution Time for Input2:
Add, s1, o1, R: 6.16908073425293e-06s
Add, s2, o1, W: 3.961846232414246e-06s
Query, s1, o1, R: 2.2724270820617676e-06s
Leak, o1, o2: 5.976296961307526e-06s

Input2:
Add, s1, o1, R
Add, s2, o1, W
Add, s1, o2, W
Query, s1, o1, W

Add, s3, o2, R
Add, s3, o3, W
Query, s1, o1, R
Leak, o1, o2
Leak, o1, o3

Execution Time for Input2:
Add, s1, o1, R: 6.2575563788414e-06s
Add, s2, o1, W: 3.6312267184257507e-06s
Add, s1, o2, W: 3.2177194952964783e-06s
Query, s1, o1, W: 2.203509211540222e-06s
Add, s3, o2, R: 3.213062882423401e-06s
Add, s3, o3, W: 2.6496127247810364e-06s
Query, s1, o1, R: 1.8132850527763367e-06s
Leak, o1, o2: 6.927177309989929e-06s
Leak, o1, o3: 1.1683441698551178e-05s

## Explanation

These statistics are obtained by adding breakpoints in the code for each type of operation.