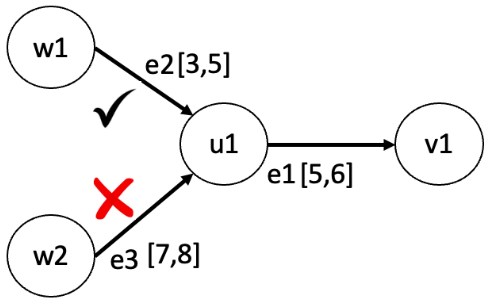
# CSE 545: Software Security
# Project Part #2

1. Write a parser for parsing sysdig output logs, and output the correctly parsed information line by line in the report. (30 points)
   a. The parser can extract the values of different fields from the log entries correctly
   b. The parser should construct 3-tuples <subject, operation, object> to represent each log entry (in Java or Python)
      - Subject with unique identifiers: process entity
        - Process unique identifier: PID, process name
      - Operation: system call operations, such as read, write, send, receive
      - Object with unique identifiers: process entity, files, IP addresses
        - Process unique identifier: PID, process name
        - File unique identifier: file name
        - IP address unique identifier: source IP, source port, destination IP, destination port, protocol

2. Format the parsed objects as a graph, and export the graph to standard format. (40 points)
   a. In the graph, nodes represent system entities (process, files, and IP addresses) and edges represent events among system entities and the directions of the edges indicate information flows.
      - E.g., <p1, write, f1> => p1 -> f1, <p1,read,f1> => p1 <- f1
   b. Connect the tuples constructed from question 1 via matching the entities
      - E.g., Given two tuples <p1, read, f1> and <p1, write, f2> that represent two edges, we can connect these two edges using p1
   c. Output the graph using DOT language
      - You may use jGraphT in Java to do so
   d. Convert output graph as images
      - Install Graphviz ([https://graphviz.org/](https://graphviz.org/))
      - Run dot -Tsvg output.dot > output.svg to show images

3. Implement backtracking algorithm (30 points)
    a. Given a point-of-interest (POI) event, perform backward graph search based on the edge directions (10 points)
        - E.g, Given p1 -> f1 as a POI event, start the search from the incoming events for p1
    b. Filter out edges based on time windows




- Backward trackability check:
    - Given an edge e1(u1, v1) for backward search and e2(w1,u1) is an incoming edge for u1, we can track from e1 to u1 iff starttime(e2) < endtime(e1)
        a. In the figure above, e1's start time is 5 and e1' endtime is 6
        b. In the figure above, e2's start time is 3 and e2 can be tracked
        c. In the figure above, e3's start time is 7 and e3 cannot be tracked
- Filter out edges based on trackability
    - For a node u to be backtrack tracked, compute the latest time from all incoming edges to u, denoted as maxEndtime(u).
    - For each incoming edge, if the start time of the edge is smaller than the maxEndtime(u), the edge should be included for further backward search
- Continue to perform the backward based on trackability until no more edges are found
- Output the graph formed by these found edges as a DOT graph

More readings:

- [https://pdos.csail.mit.edu/archive/6.824-2005/papers/king03.pdf](https://pdos.csail.mit.edu/archive/6.824-2005/papers/king03.pdf)
- https://xusheng-xiao.github.io/papers/reduction-ccs.pdf