From previous class, we have seen two different sections in the paper.

1. **Model Construction**: Three major approaches are
   a) from source (binaries), from execution traces and manual construction.
   b) from

2. **Model**: Set model, n-gram model, Control Flow Automaton.

3. **Control Flow Automaton**: Let us consider the function calls and analyze.

        main(...)               log(....)
       {                           {                          
       setuid(1000);               write(....)
       system(...);                    
       log(...);
       setuid(0);
       log(...);
    }

4. Looking at the code, it is clearly known that it is not possible to execute system(..) when uid is 0. Now, let us build the model.

![Diagram of Control Flow Automaton](attachment:diagram.png)
5. Now, even though the code never calls `system(..)` when uid is 0, our above model allows it and therefore the attacker can exploit this situation.
6. Therefore, we can use Push Down Automaton as a solution.

**Push Down Automaton (PDA):**

1. PDA on input:
   a. Change state.
   b. Check and pop top of the stack.
   c. Push new material on to the stack.
2. It looks like

   ![Diagram](attachment:image.png)

   where $\alpha = \text{input}$, $F = \text{top of stack}$ and $G = \text{pushed on to the stack}$.

3. Now, the model looks like

   ![Diagram](attachment:image.png)

4. State of this machine is kept record of using a stack for the program. As the program runs, the information is kept track using the stack. The stack looks like the following after each state.
5. Possible enhancements are
   a. To model if's better.
   b. To model arguments better.
   c. To model functions better.

6. Also, in some cases it is difficult to know which branch the program takes. For example,
   foo(…)
   {
     if(…)
       func1(…);
     else()
       {
         func2(…);
         return();
       }
     return();
   }
7. Thus, there is no good way to determine a Push Down Automaton and that’s the challenge in creating a model.
   a. The number of states we keep track can grow exponentially (multithreaded)
   b. The state of the stack represents a finite state machine and thus a regular expression. The stack PDA is deterministic even if the regular expression is infinity.
   c. Thus, the model can be rewritten so that we can get more information from the model.
      
      ```
      setuid(1000);
      null(2);
      log();
      null(-2);
      system();
      setuid(0);
      null(5);
      log();
      null(-5);
      ```
   d. The logs could not be viewed and tracked earlier. But now the null calls show the state before and after a call. Therefore, the attacker now cannot execute `system()`. 