System Call Monitor Based Intrusion Detection Systems aka Host Based IDS

- Attacker takes over execution mostly using system calls. Attacker must make sys calls to cause damage.
- Idea: Monitor System Calls made by application to detect any deviation from expected behavior

Monitor needs *Model* which describes normal behavior of application. We need to have different models for each application. Eg. Model for "cp"

{open, read, write, close}

Monitor checks for each sys call made during execution to be in this set.

Copying in chunks
open open (read write)* close close

But this has constraints.
Q: What should models be then?
A: Based on set?
   Based on regular expression? Fine state automaton?

**Incorporate Ordering Constraints**

model = FSA
Q: How do we construct models?
   - Manually
   - Tracebased
   - Statistically

**Tracebased Model Construction**
- run the application in many different ways
- run traces of system calls such that we get enough generalized traces of the run
- construct n-gram model:
  state = last n – 1 system calls made by the app
  e.g say the calls went: oorwrwrc
     oorwrrwrwrwrc
     oorwrrwrwrwcc
     oorwrrwrwrwrwrwrrwrc
     oorwrrwrwrwrwrwrwcc
  take the last two sycalls at any point of time, it can be: oo, or, rw, wr, wc, cc.

**Issue:**
- can we have a 4-gram model, what should n be ?
- how much trace data is needed
- how about having a fuzzy model instead of rigid model to minimize the dataset
- false alarms if there are not enough training data

A model can be broken by a mimicry attack: just issue system calls in the same sequence as the model is expecting.
Assuming of course that the attacker has access to the model. Or the sequence can be obtained just by tracing an app and learning the model

**Static Model Construction**

Input a source code and binary to a model constructor to get a model.
The model doesn’t compile or run the code. It analyses the input to generate the model

For copy:
main() {
    [A] open();
    [B] open();
    [C] do {
        read();
        [D] write();
        [E]
    } while (..);
    [F] close();
    [G] close();
    [H]
}

Let's compute a control flow graph: Control Flow Automation
In a CFG, each statement is a node.

open();
if (x>0) {
    read();
}
else {
    write();
}
close();

func1()
{
    .
    .
    .
    func2()
    .
    .
    .
}