Host-Based Intrusion Detection System

- Apps have some “normal” behavior
- Apps under attack will behave abnormally.

Normal Behavior

- What is “normal”?
  - Normal = expected
  - Example
    - while (1){
    - if (c==0)
    - getid();
    - else write();
  - only the system calls mentioned is called : write

Behavior

- system calls made by app
  - why system calls:
    - Attacker must use syccalls to cause damage
    - Application has to transfer control into OS regardless, so it's a convenient point of mediation

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Model Construction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Automatic from source</td>
</tr>
<tr>
<td>No/Low FPS (false positive)</td>
<td>Automatic from trace</td>
</tr>
<tr>
<td>No/Low FNS (false negative)</td>
<td>Automatic from binary</td>
</tr>
</tbody>
</table>

Better model: order of Sys calls

- N-gram model
  - $M = \{( , ) , ( , )\}$: a sequence of n system calls that the application might perform
  - Example:
    - Int main (){
    - Int fd = open ()
    - If (fd >= 0)
      - Read ()
    - Else
      - Write ()
    - Fork()
    - Close ()
    - (open,read)
    - (open,write)
    - (fork,close)
    - (read,fork)
    - (write,fork)
- Problem: we can have loops in state transition that doesn’t exist in the code, for example if we added another open after the fork, we can have an infinite loop of commands that doesn’t exist in the code
- We can use a state machine to represent the states, but that doesn’t solve the problem

Model Checking

- OS remember the state
- We can make a deterministic automata out of the non-deterministic automata
- Control flow
  - We can analyze the source code
  - We can analyze binary
  - We can’t build from system calls trace.

```
open

read

write

fork

fork

close
```