Un-trusted Code

- System call monitoring
- Inline Reference Monitors

**External Reference Monitor**

- Untrusted App
- Monitor
- Resource

**IRM**

- Untrusted Code
- Access Checks
- Resource

Monitorcheck(open,fname,mode);
Open (fname,mode);

**Advantages**
- Performance
- Fine grained controls

**Disadvantages**
- “indirect jumps”
- Complexity
- Must defend self
- Self modifying code
- Buffer over flow
Who inserts checks?

- Programmer → not trusted
- Compiler → not trusted
- User

The first option is easier and has better performance. The alternative is more complex and slower.
Monitor for memory reads and writes

<table>
<thead>
<tr>
<th></th>
<th>Un-trusted Data</th>
<th>Un-trusted Text</th>
<th>Trusted Data</th>
<th>Trusted Text</th>
</tr>
</thead>
</table>

The top adder is above the un-trusted code and the base adder is under the un-trusted text.

**Safety Checks**

cmp %r1, BaseAdder
blt abort
cmp %r1, TopAdder
bgt abort
ld %r0, [%r1]

**Safety Forcing**

and %r1, (1< L) - 1
or %r1, BaseAdder
ld %r0, [%r1]

Masked address forces it to be inbounds

%r1    x  |  y
and    0  |  y
or    BaseAdder  |  y

**Native Client**

- Break code into 32 byte chunks and all jumps are to start at chunks
- All monitored operations are in the same chunk as checks and instructions
- Cannot cross chunks