Untrusted Code (November 12, 2010)

Two main approaches:

- System Call monitoring
- Inline reference monitoring

Inline Reference Monitor:

```
monitor_check(OPEN, fname, mode);
open(fname, mode);
```

But there is a possible problem:

```
goto SNEAKY;
monitor_check(OPEN, fname, mode);
SNEAKY:
open(fname, mode);
```

Advantages
- Performance
- Fine-grained Control

Disadvantages
- Complexity
- Must defend itself (from self-modifying code, buffer overflows, etc)
Question: Who inserts checks? They can be inserted by the programmer, compiler, or the user.

Monitor for memory reads and writes

<table>
<thead>
<tr>
<th>UNTRUSTED DATA</th>
<th>top_addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNTRUSTED TEXT</td>
<td>base_addr</td>
</tr>
<tr>
<td>TRUSTED DATA</td>
<td>These trusted sections must be protected!</td>
</tr>
<tr>
<td>TRUSTED TEXT</td>
<td></td>
</tr>
</tbody>
</table>

Safety check (5 times slower):

```assembly
    cmp   %r1, base_addr
    blt abort
    cmp   %r1, top_addr
    bgt abort
    ld    %r0, [%r1]
```

Safety forcing:

```assembly
    and   %r1, ((1<<1)-1)
    or    %r1, base_addr
    ld    %r0, [%r1]
```
Problem: what about relative jumps?

Native Client solution: Break code into 32 byte chunks, and all jumps are to start of chunks.

- All monitored operations are in the same chunk as checks
- Instructions cannot cross chunks.

Safety forcing for jumps:

```assembly
and %r1, ((1<<l)-32)
or %r1, base_addr
jmp %r0
```