HexSafe: Efficient Memory Safety For C
Nathan Burow, Derrick McKee, Mathias Payer

Problem Statement
- MLOC of C/C++ in critical systems
- C/C++ have no security checks
- Constant stream of exploits:
  - Heartbleed
  - Data breaches
  - APT
- Underlying problem: Programmers don’t enforce Memory Safety

Our Approach
- Use LLVM to insert missing security checks
  - Call our runtime to validate bounds
- New hybrid metadata approach that leverages 64 bit architectures:
  - 48 of 64 bits used for virtual addresses
  - Store an ID in the unused 16 bits
  - ID is index into our metadata table
- Advantages:
  - Faster metadata look up
  - IDs propagate naturally with pointers

```
void main() {
    char buf[10];
    while((c = getc()) != ' \n') {
        buf[i++] = c;
    }
}
```

Instrumentation

```
void main() {
    char buf[10];
    _memsafe.instrument(buf, 10)
    while((c = getc()) != ' \n') {
        _memsafe.check(buf + i);
        buf[i++] = c;
    }
}
```

<table>
<thead>
<tr>
<th>ID</th>
<th>Base</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xdeadbeef</td>
<td>10</td>
</tr>
</tbody>
</table>

$$\text{base} \leq \text{buf + i} < \text{base + length}$$