Automatic Memory-Leak Fixing for C Programs

Qing Gao, Yingfei Xiong, Lu Zhang, Bing Xie, Hong Mei
Peking University
2013-12-21
Memory Leak

- A program consumes memory but is unable to release it.

```c
int f(int num, int len)
{
    int *array = (int*)malloc(sizeof(int)*len);
    if (array == NULL) return -1;
    //operation on array

    int sum = 0;
    if (num < 0) free(array);
    else
    {
        for (int i = 0; i < len; i++)
            if (array[i] != num)
                sum += array[i];
        else
            return 0;
    }

    printf("%d", sum);
    return sum;
}
```
Memory leak [Clause and Orso, 2010]

- Difficult to detect by testing
- Potential to impact multiple applications
- Common even in mature applications
Outline

1. Memory Leak
2. Existing work
3. Approach
4. Experiments
5. Discussion
Existing work—Memory leak detection

- Static detection:
  - False positives

- Dynamic detection
  - Runtime overhead

Programmers need to check programs to fix leaks
  - Need some time
  - Correctness is not ensured
Existing work—
Dynamic pinpointing

- Need to run program
- Instrument nearly every instruction and overhead is high
- Programmers still need to check programs
Existing work—Compile-time object deallocation

- Focus on Java bytecode
  - May lead to double free
  - May lead to unreadable code
  - Cannot make sure whether a leak has been completely fixed
Outline

1. Memory Leak
2. Existing work
3. Approach
4. Experiments
5. Discussion
Correct fix definition

- A location in the code to insert “free(p)” where for any path covers this location
  - Pointer p points to an allocated chunk at the location (*allocation and its reference*)
  - There is no other deallocation statements that release the memory (*no double frees*)
  - There is no reference to the memory after the execution (*no use after free*)
Approach overview

- Building A-CFG
- Memory leak detection
- Memory Leak fixing
- Map A-CFG to code
Building A-CFG

- A-CFG: Allocation-centric Control Flow Graph

- Transformed from CFG (control flow graph)
  - Have no cycles (no loops)
  - Abstract information only related to memory management
Building A–CFG

- **Input:** CFG

```
array = ((int *)malloc(sizeof(int) * len));

switch (array == ((void *)0))
  
  0

  sum = 0;

  return -1;

  switch (num < 0)
    
    0

    i = 0;

    (free(array));

    switch (i < len)
      
      0

      (printf("%d", sum));

      return sum;

      return 0;

      sum = (sum + (array[i]));

      i = (i + 1);
```

Exit
Building A-CFG

- Output: A-CFG

- Node type:
  - Normal
  - Allocation

- Node label:
  - Dealloc: definite/possible
  - Ref: possible

- Edge label: pointer set
Building A–CFG

- Inter-procedure analysis
  - Build procedure summaries
    - Method $m$ with parameters $p_0, \ldots, p_n$

\[
\langle S_0, F_0 \rangle \times \ldots \times \langle S_n, F_n \rangle \rightarrow \langle S_r \rangle
\]

- $S_j$: what $*p_j$ will point to
- $F_j$: whether $p_j$ is freed
  - May and Must summaries
- Map summaries to A-CFG nodes
Memory Leak Detection

- Classify the edges by deallocation:
  - Blue: definitely reachable
  - Yellow: possibly reachable
  - Red: unreachable
Memory Leak Detection

- Classify the edges by deallocation:
  - Blue: definitely reachable
  - Yellow: possibly reachable
  - Red: unreachable
Memory leak fixing

- Fix on edges whose pointer set is not empty:
  - allocation and its references

- Fix on red edges:
  - no double free

- Traverse backwardly:
  - no use after free
Outline

1. Memory Leak
2. Existing work
3. Approach
4. Experiments
5. Discussion
Experiment

- RQ1: How effective is our tool in fixing real-world memory leaks?

- RQ2: What are the execution time and memory consumption of our tool for memory leak fixing?
Experiment

- **Benchmark: SPEC2000**
  - Used by papers related to detection

<table>
<thead>
<tr>
<th>Programs</th>
<th>Size (Kloc)</th>
<th>#Func</th>
<th>#Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>1.3</td>
<td>44</td>
<td>11</td>
</tr>
<tr>
<td>equake</td>
<td>1.5</td>
<td>45</td>
<td>29</td>
</tr>
<tr>
<td>mcf</td>
<td>1.9</td>
<td>44</td>
<td>3</td>
</tr>
<tr>
<td>bzip2</td>
<td>4.6</td>
<td>92</td>
<td>10</td>
</tr>
<tr>
<td>gzip</td>
<td>7.8</td>
<td>128</td>
<td>5</td>
</tr>
<tr>
<td>parser</td>
<td>10.9</td>
<td>342</td>
<td>1</td>
</tr>
<tr>
<td>ammp</td>
<td>13.3</td>
<td>197</td>
<td>37</td>
</tr>
<tr>
<td>vpr</td>
<td>17</td>
<td>290</td>
<td>2</td>
</tr>
<tr>
<td>crafty</td>
<td>18.9</td>
<td>127</td>
<td>12</td>
</tr>
<tr>
<td>twolf</td>
<td>19.7</td>
<td>209</td>
<td>2</td>
</tr>
<tr>
<td>mesa</td>
<td>49.7</td>
<td>1124</td>
<td>67</td>
</tr>
<tr>
<td>vortex</td>
<td>52.7</td>
<td>941</td>
<td>8</td>
</tr>
<tr>
<td>gap</td>
<td>59.5</td>
<td>872</td>
<td>2</td>
</tr>
<tr>
<td>gcc</td>
<td>205.8</td>
<td>2271</td>
<td>53</td>
</tr>
</tbody>
</table>
## Experimental Results

### Effectiveness

<table>
<thead>
<tr>
<th>Programs</th>
<th>#Fixed</th>
<th>#Maximum Detected</th>
<th>Percentage(%)</th>
<th>#Fixing Points</th>
<th>#Unnecessary Fixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>equake</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mcf</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bzip2</td>
<td>1(1,1)</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>gzip</td>
<td>1(1,1)</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>parser</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ammp</td>
<td>15(7,15)</td>
<td>20</td>
<td>75</td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>vpr</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>crafty</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>twolf</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mesa</td>
<td>1(0,0)</td>
<td>9</td>
<td>11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>vortex</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>gap</td>
<td>0</td>
<td>0</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>gcc</td>
<td>7(1,2)</td>
<td>44</td>
<td>16</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td>25(10,19)</td>
<td>85</td>
<td>29</td>
<td>40</td>
<td>0</td>
</tr>
</tbody>
</table>
Leaks reported by detection tools

<table>
<thead>
<tr>
<th>Programs</th>
<th>LC</th>
<th>Fastcheck</th>
<th>SPARROW</th>
<th>SABER</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>1(0)</td>
<td>1(0)</td>
<td>1(0)</td>
<td>1(0)</td>
</tr>
<tr>
<td>equake</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
</tr>
<tr>
<td>mcf</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
</tr>
<tr>
<td>bzip2</td>
<td>1(1)</td>
<td>0(0)</td>
<td>1(0)</td>
<td>1(0)</td>
</tr>
<tr>
<td>gzip</td>
<td>1(2)</td>
<td>0(0)</td>
<td>1(4)</td>
<td>1(0)</td>
</tr>
<tr>
<td>parser</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
</tr>
<tr>
<td>ammp</td>
<td>20(4)</td>
<td>20(0)</td>
<td>20(0)</td>
<td>20(0)</td>
</tr>
<tr>
<td>vpr</td>
<td>0(0)</td>
<td>0(1)</td>
<td>0(9)</td>
<td>0(3)</td>
</tr>
<tr>
<td>crafty</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
</tr>
<tr>
<td>twolf</td>
<td>0(0)</td>
<td>2(0)</td>
<td>5(0)</td>
<td>5(0)</td>
</tr>
<tr>
<td>mesa</td>
<td>2(0)</td>
<td>0(2)</td>
<td>9(0)</td>
<td>7(4)</td>
</tr>
<tr>
<td>vortex</td>
<td>0(26)</td>
<td>0(0)</td>
<td>0(1)</td>
<td>0(4)</td>
</tr>
<tr>
<td>gap</td>
<td>0(1)</td>
<td>0(0)</td>
<td>0(0)</td>
<td>0(0)</td>
</tr>
<tr>
<td>gcc</td>
<td>N/A</td>
<td>35(2)</td>
<td>44(1)</td>
<td>40(5)</td>
</tr>
<tr>
<td>total</td>
<td>25(34)</td>
<td>58(5)</td>
<td>81(15)</td>
<td>70(14)</td>
</tr>
</tbody>
</table>
## Experimental Results

### Efficiency

<table>
<thead>
<tr>
<th>Programs</th>
<th>Size (Kloc)</th>
<th>Crystal Time (sec)</th>
<th>LeakFix Time (sec)</th>
<th>Total Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Preprocessed Code Parsing</td>
<td>CFG Building</td>
<td>A-CFG Building</td>
</tr>
<tr>
<td>art</td>
<td>1.3</td>
<td>0.187</td>
<td>0.034</td>
<td>0.031</td>
</tr>
<tr>
<td>equake</td>
<td>1.5</td>
<td>0.219</td>
<td>0.068</td>
<td>0.072</td>
</tr>
<tr>
<td>mcf</td>
<td>1.9</td>
<td>1.610</td>
<td>0.257</td>
<td>0.045</td>
</tr>
<tr>
<td>bzip2</td>
<td>4.6</td>
<td>0.424</td>
<td>0.109</td>
<td>0.097</td>
</tr>
<tr>
<td>gzip</td>
<td>7.8</td>
<td>2.104</td>
<td>0.498</td>
<td>0.276</td>
</tr>
<tr>
<td>parser</td>
<td>10.9</td>
<td>2.653</td>
<td>0.561</td>
<td>0.298</td>
</tr>
<tr>
<td>ammp</td>
<td>13.3</td>
<td>4.415</td>
<td>0.989</td>
<td>0.351</td>
</tr>
<tr>
<td>vpr</td>
<td>17</td>
<td>3.32</td>
<td>0.675</td>
<td>0.498</td>
</tr>
<tr>
<td>crafty</td>
<td>18.9</td>
<td>6.177</td>
<td>1.184</td>
<td>0.604</td>
</tr>
<tr>
<td>twolf</td>
<td>19.7</td>
<td>10.106</td>
<td>1.995</td>
<td>1.012</td>
</tr>
<tr>
<td>mesa</td>
<td>49.7</td>
<td>16.948</td>
<td>2.312</td>
<td>3.608</td>
</tr>
<tr>
<td>vortex</td>
<td>52.7</td>
<td>14.995</td>
<td>2.555</td>
<td>1.721</td>
</tr>
<tr>
<td>gap</td>
<td>59.5</td>
<td>8.201</td>
<td>2.170</td>
<td>1.325</td>
</tr>
<tr>
<td>gcc</td>
<td>205.8</td>
<td>18.701</td>
<td>4.669</td>
<td>4.973</td>
</tr>
</tbody>
</table>
Experimental Results

Efficiency (cont.)

<table>
<thead>
<tr>
<th>Programs</th>
<th>Size (Kloc)</th>
<th>Percentage(%)</th>
<th>Maximum Memory (Megabyte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>1.3</td>
<td>12.6</td>
<td>11.2</td>
</tr>
<tr>
<td>equake</td>
<td>1.5</td>
<td>20.0</td>
<td>10.6</td>
</tr>
<tr>
<td>mcf</td>
<td>1.9</td>
<td>2.4</td>
<td>8.4</td>
</tr>
<tr>
<td>bzip2</td>
<td>4.6</td>
<td>15.8</td>
<td>13.2</td>
</tr>
<tr>
<td>gzip</td>
<td>7.8</td>
<td>9.7</td>
<td>11.4</td>
</tr>
<tr>
<td>parser</td>
<td>10.9</td>
<td>8.6</td>
<td>13.8</td>
</tr>
<tr>
<td>ammp</td>
<td>13.3</td>
<td>6.2</td>
<td>15.3</td>
</tr>
<tr>
<td>vpr</td>
<td>17</td>
<td>11.2</td>
<td>18.2</td>
</tr>
<tr>
<td>crafty</td>
<td>18.9</td>
<td>7.7</td>
<td>21.7</td>
</tr>
<tr>
<td>twolf</td>
<td>19.7</td>
<td>7.8</td>
<td>19.7</td>
</tr>
<tr>
<td>mesa</td>
<td>49.7</td>
<td>16.5</td>
<td>34.9</td>
</tr>
<tr>
<td>vortex</td>
<td>52.7</td>
<td>9.0</td>
<td>40.5</td>
</tr>
<tr>
<td>gap</td>
<td>59.5</td>
<td>11.4</td>
<td>38.5</td>
</tr>
<tr>
<td>gcc</td>
<td>205.8</td>
<td>19.3</td>
<td>79.3</td>
</tr>
</tbody>
</table>
Example bugs

63: int u_v_nonbon( V, lambda )
  float *V.lambda;
65: {
...
103:  buffer = malloc( 3*i*sizeof(float) );
104:  if( buffer == NULL )
105:     { aaerror("cannot allocate memory in u_v_nonbon\n"); return 0; }
106:  vector = malloc( i*sizeof(float) );
107:  if( vector == NULL )
108:     { aaerror("cannot allocate memory in u_v_nonbon\n"); return 0; }
109:  atms = malloc( i*sizeof(ATOM *) );
110:  if( atms == NULL )
111:     { aaerror("cannot allocate memory in u_v_nonbon\n"); return 0; }
...
118:  for( jj=1; jj<imax; jj++,a1=bp ){
...
122:     if( (*use)[used] == a1 )
123:         { used += 1; }
124:     else { aaerror("error in uselist - must abort"); return 0; }
...
178:  }
...  
183: }
Example bugs

319: struct gl_texture_image *gl_alloc_texture_image( void )
320: {
321:     return (struct gl_texture_image*) calloc(1, sizeof(struct gl_texture_image));
322: }

344: static struct gl_texture_image *
      image_to_texture( GLcontext *ctx, const struct gl_image *image,
                      GLenum internalFormat, GLint border )
347: {
349:     struct gl_texture_image *texImage;
...
362:     texImage = gl_alloc_texture_image();
...
451:     switch (texImage->Format) {
...
476:         default:
477:             gl_problem(ctx,"Bad format in image_to_texture");
478:             return NULL;
479:         }
...
787: }
Outline

1. Memory Leak
2. Existing work
3. Approach
4. Experiments
5. Discussion
Discussion

- Handling loops
  - Loops are handled independently
- Unnecessary fixes
- Related Allocations
  - free(p->q) v.s. free(p)
- Minimal number of fixes

```c
if (condition1)
    statement1;
else
    statement2;
statement3;
```
Thanks!