Lightweight and Modular Resource Leak Verification

Martin Kellogg\textsuperscript{a}, Narges Shadab\textsuperscript{b}, Manu Sridharan\textsuperscript{b}, Michael D. Ernst\textsuperscript{a}

\textsuperscript{a}University of Washington \hspace{1cm} \textsuperscript{b}University of California, Riverside
What’s a Resource Leak?

```java
try {
    Socket s = new Socket(address, port);
    ...
    s.close();
} catch (IOException e) {
}
```
What’s a Resource Leak?

```java
try {
    Socket s = new Socket(address, port);
    ...
}
```

Missing call to close()
Problems Caused by Resource Leaks

- Resource starvation
- Slowdowns
- System crashes
- Denial-of-service attack
  - E.g. CVE-1999-1127, CVE-2001-0830, CVE-2002-1372
Key Challenge: Pointer Aliasing

- Resource can be closed through *any* alias
Key Challenge: Pointer Aliasing

- Resource can be closed through *any* alias
- Previous approaches:
Key Challenge: Pointer Aliasing

- Resource can be closed through *any* alias
- Previous approaches:

  * Heuristic bug-finding tools
  * Ignore aliasing
Key Challenge: Pointer Aliasing

- Resource can be closed through *any* alias
- Previous approaches:

<table>
<thead>
<tr>
<th>Heuristic bug-finding tools</th>
<th>Ownership type systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignore aliasing</td>
<td>Enforce uniqueness</td>
</tr>
</tbody>
</table>
Key Challenge: Pointer Aliasing

- Resource can be closed through *any* alias
- Previous approaches:

<table>
<thead>
<tr>
<th>Heuristic bug-finding tools</th>
<th>Ownership type systems</th>
<th>Whole-program static analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignore aliasing</td>
<td>Enforce uniqueness</td>
<td>Track all aliases</td>
</tr>
</tbody>
</table>
Key Insight

- Resource leak detection is an accumulation problem
Key Insight

- Resource leak detection is an **accumulation** problem
  - FSM contains no loops
Key Insight

- Resource leak detection is an **accumulation** problem
  - FSM contains no loops
  - **Sound with no alias analysis**
Resource Leaks as Accumulation
Resource Leaks as Accumulation

Going out of scope

Open

Going out of scope

Error

Exit

closed
Resource Leaks as Accumulation

FSM contains no loops
Resource Leaks as Accumulation

FSM contains no loops

Alias analysis not required for soundness
Resource Leaks as Accumulation

- Open → closed (close())
- Going out of scope

- closed → Error
- Going out of scope

- closed → closed (close())

- FSM contains no loops
- Alias analysis not required for soundness
- can be implemented modularly
Leak Detection Approach:

1. Compute what methods must be called
2. Compute what methods are called
3. Issue error if mismatch when going out of scope
Example

{
    s = new Socket(address, port);
    ...
    if (...) {
        s = ...
    }
    s.close();
}
Example

```
{
    s = new Socket(address, port);
    ...
    if (...) {
        s = ...
    }
    s.close();
}
```

**Obligation:** call close on s
Example

```java
{s = new Socket(address, port);
...
if (...) {
    s = ...
}
}
s.close();
```

**Obligation:** call `close` on `s`

**Called Methods:** `{}`

**Called Methods:** `{"close"}`
Example

```java
{ 
    s = new Socket(address, port);  // Obligation: call close on s
    ...
    if (...) {
        s = ...;  // Error
    }
    s.close();  // Called Methods: {}"close"
}
```
Example

```java
{ 
    s = new Socket(address, port);  // Obligation: call close on s

    ... 

    if (...) {
        s = ...;
    }

    s.close();  // Called Methods: {"close"}
}
```
Example

```java
{s = new Socket(address, port);
 ...
if (...) {
    s = ...;
}

   t = s;
   t.close();
}
```

**Obligation:** call `close` on `s`

**Called Methods:** {}

**Error**
Example

```java
{s = new Socket(address, port);
...
if (...) {
    s = ...;
}
}
t = s;
\textbf{\textcolor{blue}{t.close();}}
```

Obligation: call \texttt{close} on \texttt{s}

Called Methods: \{\}

Error

\textbf{\textcolor{blue}{Sound but not precise}}

Called Methods: \{\}

32
Precision via Local Alias Reasoning

- Local must-aliases
- Lightweight ownership
- Resource aliasing
- Obligation creation
Precision via Local Alias Reasoning

- Local must-aliases
- Lightweight ownership
- Resource aliasing
- Obligation creation
Lightweight Ownership

closeSocket(mySock);
Lightweight Ownership

**Obligation:** call close on mySock

closeSocket(mySock);

**Obligation:** call close on mySock
Lightweight Ownership

**Obligation:** call close on mySock

closeSocket(mySock);

**Obligation:** call close on mySock

```java
void closeSocket(@Owning Socket s) {
    s.close();
}
```

**Obligation:** call close on s

```java
s.close();
```
Lightweight Ownership

**Obligation:** call `close` on `mySock`

```java
void closeSocket(@Owning Socket s) {
    Obligation: call `close` on `s`
    s.close();
}
```

- Obligations are neither created nor destroyed
- Doesn’t restrict privileges of other aliases
Precision via Local Alias Reasoning

- Local must-aliases
- Lightweight ownership
- Resource aliasing
- Obligation creation
Resource Aliasing

Socket socket = ...;
InputStreamReader stream =
    new InputStreamReader(socket.getInputStream());
...

Socket socket = ...;
InputStreamReader stream =
    new InputStreamReader(socket.getInputStream());
...

Which of these should be closed?
Resource Aliasing

Socket socket = ...;
InputStreamReader stream =
    new InputStreamReader(socket.getInputStream());
...

Which of these should be closed?

- Closing either socket or stream is adequate
- Extensibility
Evaluation:
## Evaluation: Case Studies

Four programs: zookeeper, hadoop-hdfs, hbase, plume-util

<table>
<thead>
<tr>
<th>Lines of code</th>
<th>Resource Leaks Found</th>
<th>False positive warnings</th>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>427,858</td>
<td>49</td>
<td>121</td>
<td>286</td>
</tr>
</tbody>
</table>
## Evaluation: Case Studies

### Four programs: zookeeper, hadoop-hdfs, hbase, plume-util

<table>
<thead>
<tr>
<th>Lines of code</th>
<th>Resource Leaks Found</th>
<th>False positive warnings</th>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>427,858</td>
<td>49</td>
<td>121</td>
<td>286</td>
</tr>
</tbody>
</table>

**Precision: 29%**
## Evaluation: Case Studies

Four programs: zookeeper, hadoop-hdfs, hbase, plume-util

<table>
<thead>
<tr>
<th>Lines of code</th>
<th>Resource Leaks Found</th>
<th>False positive warnings</th>
<th>Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>427,858</td>
<td>49</td>
<td>121</td>
<td>286</td>
</tr>
</tbody>
</table>

~1 per 1,500 LoC
Evaluation: Comparison

3 analyses:

- **RLC**, our type-based analysis
- **Eclipse**’s high-confidence heuristic bug-finder
- **Grapple**, a whole-program graph reachability analysis
Evaluation: Comparison

Recall

RLC

Eclipse

Grapple

100%
Evaluation: Comparison

Recall

- RLC
- Eclipse
- Grapple

Eclipse and Grapple miss most real leaks

100%
Evaluation: Comparison

Recall

- RLC
- Eclipse
- Grapple

Precision

- RLC
- Eclipse
- Grapple

100%
Evaluation: Comparison

Recall
- RLC: 100%
- Eclipse: 100%
- Grapple: 100%

Precision
- RLC: 100%
- Eclipse: 100%
- Grapple: 100%

Time
- RLC: ~37 hrs
- Eclipse: Grapple
- Grapple: Grapple
Evaluation: Comparison

**Recall**
- RLC: 100%
- Eclipse: 100%
- Grapple: 100%

**Precision**
- RLC: 100%
- Eclipse: 100%
- Grapple: 100%

**Time**
- RLC: ~37 hrs
- Eclipse: ~37 hrs
- Grapple: ~27 hrs
Evaluation: Comparison

Recall
- RLC
- Eclipse: 100%
- Grapple

Precision
- RLC
- Eclipse: 100%
- Grapple: 100%

Time
- RLC
- Eclipse
- Grapple: 1 hr
Contributions

- Lightweight and modular resource leak verification via accumulation analysis
- Local alias reasoning for precision
- Extensive evaluation
- Open-source implementation at checkerframework.org