Qualitative & Quantitative Evaluation of Static Code Analysis Tools

Indiana University-Purdue University Indianapolis
Dr. James H. Hill (hillj@cs.iupui.edu)

December 16-18, 2014
Static code analysis (SCA) is a methodology of detecting errors in program based on the review of code marked by the analyzer in areas where potential errors may occur.

SCA tools aid developers in quickly identifying errors through automation:
- memory leaks
- dead code
- code conformance
- etc.
Existing Static Code Analysis Tools

Given the vast number of SCA tools, it can be hard identifying what SCA tools are best for the job!

<table>
<thead>
<tr>
<th>HP Fortify Source Code Analyzer</th>
<th>AdaControl</th>
<th>Pylint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>Astrée</td>
<td>Parasoft C/C++test</td>
</tr>
<tr>
<td>IBM Rational AppScan Source Edition</td>
<td>cpplint</td>
<td>Klocwork Insight</td>
</tr>
<tr>
<td>Imagix 4D</td>
<td>Clang</td>
<td>SofCheck Inspector</td>
</tr>
<tr>
<td>MALPAS</td>
<td>PVS-Studio</td>
<td>CodeRush</td>
</tr>
<tr>
<td>CodeSonar</td>
<td>Cppcheck</td>
<td>Visual Studio Team System</td>
</tr>
<tr>
<td>CodElt.Right</td>
<td>Protecode</td>
<td>DMS Software Reengineering Toolkit</td>
</tr>
<tr>
<td>FxCop</td>
<td>FindBugs</td>
<td>Kalistick</td>
</tr>
<tr>
<td>Apparat</td>
<td>PMD</td>
<td>...</td>
</tr>
</tbody>
</table>

It is also a costly & time-consuming process evaluating the quality of each tool...

Our objective is to evaluate the quality of static code analysis tool, and understand how to best apply them to a given piece of source code.
Current Status

• Acquired and deployed three commercial SCA tools into the System Integration Lab at IUPUI
• Developed an extensible framework for automating the evaluation of SCA tools (SCATE)
• Exploring methodology and reporting features
  – Granularity
  – Aggregating multiple tools
  – Permutation heat map
Granularity

- Controls the required accuracy for the tool
Granularity

- Controls the required accuracy for the tool

```c
void CWE835_Infinite_Loop__do_01_bad() {
    int i = 0;
    /* FLAW: Infinite Loop - do..while() with no break point */
    do {
        printIntLine(i);
        i = (i + 1) % 256;
    } while(i >= 0);
}
```
Granularity

- Controls the required accuracy for the tool

**CWE835_Infinite_Loop__do_01.c**

```c
void CWE835_Infinite_Loop__do_01_bad() {
    int i = 0;

    /* FLAW: Infinite Loop - do..while() with no break point */
    do {
        printIntLine(i);
        i = (i + 1) % 256;
    } while(i >= 0);
}
```

**Flaw**
Granularity

- Controls the required accuracy for the tool

CWE835_Infinite_Loop__do_01.c

```c
10 void CWE835_Infinite_Loop__do_01_bad() {
11     int i = 0;
12
13     /* FLAW: Infinite Loop - do..while() with no break point */
14     do
15     {
16         printIntLine(i);
17         i = (i + 1) % 256;
18     } while(i >= 0);
19 }
```

**FILE**
The tool can find the flaw anywhere in the file

**FUNCTION**
The tool can find the flaw anywhere in the function

**LINE**
The tool must find the flaw on line 14
Granularity

• Controls the required accuracy for the tool

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Detected Flaws</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>25,511</td>
</tr>
<tr>
<td>Function</td>
<td>3,565</td>
</tr>
<tr>
<td>Line</td>
<td>2,215</td>
</tr>
</tbody>
</table>

• Increasing granularity reduces the quality of a Tool
Using Multiple Tools

- Organizations will often run multiple tools to reduce risk
Using Multiple Tools

- Organizations will often run multiple tools to reduce risk

<table>
<thead>
<tr>
<th>Tool 1</th>
<th>Tool 2</th>
<th>Tool 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>FP</td>
<td>TP</td>
</tr>
<tr>
<td>5,155</td>
<td>206,433</td>
<td>12,235</td>
</tr>
</tbody>
</table>
Using Multiple Tools

- Organizations will often run multiple tools to reduce risk

<table>
<thead>
<tr>
<th>Tool 1</th>
<th>Tool 2</th>
<th>Tool 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>FP</td>
<td>TP</td>
</tr>
<tr>
<td>5,155</td>
<td>206,433</td>
<td>12,235</td>
</tr>
</tbody>
</table>

Aggregate:

<table>
<thead>
<tr>
<th>TP</th>
<th>FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>31,329</td>
<td>327,004</td>
</tr>
</tbody>
</table>
Using Multiple Tools

- Organizations will often run multiple tools to reduce risk

<table>
<thead>
<tr>
<th>Tool 1</th>
<th>Tool 2</th>
<th>Tool 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>FP</td>
<td>TP</td>
</tr>
<tr>
<td>5,155</td>
<td>206,433</td>
<td>12,235</td>
</tr>
</tbody>
</table>

11,572 Duplicate TPs
Using Multiple Tools

- Organizations will often run multiple tools to reduce risk

<table>
<thead>
<tr>
<th>Tool 1</th>
<th>Tool 2</th>
<th>Tool 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>FP</td>
<td>TP</td>
</tr>
<tr>
<td>5,155</td>
<td>206,433</td>
<td>12,235</td>
</tr>
</tbody>
</table>

- Not all TPs are unique
- TPs increase but at the cost of many more FPs (reduced precision)
Permutation Heat Map

- Permutations use Data and/or Control flows to obscurify a Flaw to test SCA tools
Permutation Heat Map

- Permutations use Data and/or Control flows to obscurify a Flaw to test SCA tools

**Permutation 01:**
```c
void bad (void)
{
    // FLAW: ...
}
```

**Permutation 02:**
```c
void bad (void)
{
    if (1)
    {
        // FLAW: ...
    }
}
```

**Permutation 03:**
```c
void bad (void)
{
    if (5==5)
    {
        // FLAW: ...
    }
}
```
Permutation Heat Map
Permutation Heat Map

CWE 415 (Double Free)
15: 50% Flaws Found
16: 100% Flaws Found
Permutation Heat Map

CWE 415 (Double Free)
15: 50% Flaws Found
16: 100% Flaws Found

Permutation 15:

```c
void bad (void) {
    switch(6) {
    case (6):
        // FLAW: ...
        break
    break
    }
}
```

Permutation 16:

```c
void bad (void) {
    while (1) {
        // FLAW: ...
        break
    }
}
```
Permutation Heat Map

CWE 415 (Double Free)
15: 50% Flaws Found
16: 100% Flaws Found

**Permutation 15:**
```c
void bad (void)
{
    switch (6) {
        case (6):
            // FLAW: ...
            break
        // FLAW: ...
        break
    }
}
```

**Permutation 16:**
```c
void bad (void)
{
    while (1) {
        // FLAW: ...
        break
    }
}
```

- The type of permutation can affect a tool’s quality
Future Work

• SWAMP Integration
• Tool behavioral model
• Predict tool quality against source code
• Streamline analysis into a cloud-based testing as a service product
Future Work

- SWAMP Integration
- Tool behavioral model
- Predict tool quality against source code
- Streamline analysis into a cloud-based testing as a service product

The SWAMP has multiple SCA tools integrated into their environment and can provide tool results.
Future Work

- SWAMP Integration
- Tool behavioral model
- Predict tool quality against source code
- Streamline analysis into a cloud-based testing as a service product
Future Work

- SWAMP Integration
- Tool behavioral model
- Predict tool quality against source code
- Streamline analysis into a cloud-based testing as a service product
Future Work

• SWAMP Integration
• Tool behavioral model
• Predict tool quality against source code
• Streamline analysis into a cloud-based testing as a service product