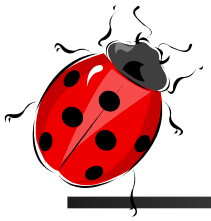

Towards Easing the Diagnosis of Bugs in OS Code

Henrik Stuart, René Rydhof Hansen, Julia Lawall
and Jesper Andersen (DIKU, Denmark)

Gilles Muller, Yoann Padioleau (EMN, France)

the Coccinelle project

Issues in bug finding/diagnosis

- Metal [OSDI 2000]
  - Control-flow based analysis
- Too many false positives
  - Requires manual investigation
  - Delays the deployment logging/workaround/prevention code

Our previous work: Collateral Evolutions

- Evolution

  in a library

lib.c

becomes

```
int foo(int x){
int bar(int x){
```

- Can entail lots of

  Collateral Evolutions in clients

Legend:

before

after

Client_1.c

```
foo(1);
bar(1);

foo(2);
bar(2);
```

Client_2.c

```
foo(foo(2));
bar(bar(2));

if(foo(3)) {
if(bar(3)) {
```

Client_n.c

SmPL: Semantic Patch Language

- Like patch code, but generic
- Abstracts away irrelevant details:
  - Differences in spacing, indentation, and comments
  - Choice of variable names (metavariables)
  - Device-specific control structure (control-flow oriented rather than AST oriented)
  - Other variations in coding style (isomorphisms)

One semantic patch can modify most, if not all, affected files.

A simple sample of SmPL

```
@@
function xxx_info;

identifier x,y;
@@
   int xxx_info(int x
 +                 ,scsi *y
                          ) {
 -    scsi *y;
      ...
 -    y = scsi_get();
 -    if(!y) { ... return -1; }
      ...
 -    scsi_put(y);
      ...
   }
```

metavariables

metavariable references

Control-flow '...' operator

modifiers

Finding bugs with Coccinelle

- Use the pattern description capabilities of SmPL to describe a bug

- WYSIWIB

  What You See Is Where It Bugs !!!

Double enabling or disabling of interrupts

```
@@  @@
 cli();
 ... WHEN != ( sti(); | restore_flags(...); )
 ? cli();

@@  @@
( sti(); |  restore_flags(...); )
 ... WHEN != cli();
( sti(); |  restore_flags(...); )
```

Calling kmalloc with interrupts disabled

@@ @@

cli();

... WHEN != (sti(); | restore_flags(...);)

kmalloc(...,GFP_KERNEL);

Fixing bugs with Coccinelle

- Introducing workaround/logging code
  - Modification capability of SmPL (+, -)
- Temporary solution until the bug is really fixed
- Sometimes corrects the problem

Warning about bugs in interrupt status management

```
@@  @@
 cli();
 ... WHEN != ( sti(); | restore_flags(...); )
+ warn("Double interrupt disable in %s", _FUNCTION__);
 cli();

@@  @@
( sti(); |  restore_flags(...); )
 ... WHEN != cli();
( sti(); |  restore_flags(...);
+ warn("Double interrupt disable in %s", __FUNCTION__);
```

Changing kmalloc flag

```
@@  @@
  cli();
  ... WHEN != ( sti(); | restore_flags(...); )
- kmalloc(e,GFP_KERNEL);
+ kmalloc(e,GFP_ATOMIC);
```

Comparison with Metal [OSDI'00]

- For interrupt checking, use of freed memory, deref of null ptrs
  - Detect 85-100% of Metal-detected bugs
  - Detect some bugs not detected by Metal
- Issues:
  - Coccinelle parses cpp code
    - A few more parse errors
    - But detect a few more bugs
  - Coccinelle is purely intra-procedural
- Comparable rate of false positives

Towards reducing false positives

- Problem: Coccinelle performs only syntactic matching
- Example:

```
@@ expression E;  @@
kfree(E);
...
E
```

Towards reducing false positives

- Problem: Coccinelle performs only syntactic matching
- Integrate data-flow information:

```
@@ expression E;  @@
kfree(E);
...
E
where E1 = E2 and E1.dfa = E2.dfa
```

Scripting, to allow more complex computations

```
@@ identifier I; expression E; constant C; type T; @@
T I[C];
<...
 I[E]
...>

@@ script: python: C as x, E as y  @@
buffer_size = cocci_lib.dfa(x).eval[1]
index_values = cocci_lib.dfa(y).eval

cocci_lib.include_match(max(index_values) >= buffer_size)
```

Assessment

- SmPL provides a WYSIWIB specification of bug conditions
- The bugs we have looked at are more generic than typical collateral evolutions
  - Need for extra semantic information
  - Need for more complex computations via a scripting interface
- This is all work in progress!