Coccinelle: A Program Matching and Transformation Tool for Linux

Nicolas Palix (DIKU)
joint work with
Julia Lawall (DIKU), Gilles Muller (INRIA)
Jesper Andersen, Julien Brunel,
René Rydhof Hansen, and Yoann Padioleau

http://coccinelle.lip6.fr/
The problem: Dealing with Linux Code

- It's huge
  - 6 MLOC
  - Increased by almost 50% in 3 years
  - Over 50% dedicated to drivers
The problem: Dealing with Linux Code

- It’s huge

- It’s configuration polymorphic
  - Several platforms
  - Many combinations of devices.
The problem: Dealing with Linux Code

- It’s huge
- It’s configuration polymorphic
- It’s (unfortunately) buggy
Bugs’ lives

- **Erroneous:**
  - Kmalloc, IsNULL, NotAnd, Find_Unsigned
- **Suspicious:** NULL_ref, NotNULL
- **Bad practices:** Unused
The problem: Dealing with Linux Code

- It’s huge
- It’s configuration polymorphic
- It’s (unfortunately) buggy
- It’s written in C
  - Error prone language
The problem: Dealing with Linux Code

- It’s huge
- It’s configuration polymorphic
- It’s (unfortunately) buggy
- It’s written in C
- It evolves continuously
Can you still follow?

Linux Kernel Development
Greg Kroah-Hartman, SuSE Labs /Novell Inc.
Jonathan Corbet, LWN.net
Amanda McPherson, The Linux Foundation
Changes in detail

Linux Kernel Development
Greg Kroah-Hartman, SuSE Labs / Novell Inc.
Jonathan Corbet, LWN.net
Amanda McPherson, The Linux Foundation
Two problems

- **Bug finding (and fixing)**
  - Search for patterns of wrong code
  - Systematically fix found wrong code

- **Collateral evolutions**
  - Evolution in a library interface entails lots of Collateral Evolutions in clients
    - Search for patterns of interaction with the library
    - Systematically transform the interaction code
The Coccinelle tool

- Program matching and transformation for unpreprocessed C code.
- Fits with the existing habits of Linux programmers.

Semantic Patch Language (SmPL):
- Based on the syntax of patches,
- Declarative approach to transformation
- High level search that abstracts away from irrelevant details
- A single small semantic patch can modify hundreds of files, at thousands of code sites
Using SmPL to abstract away from irrelevant details

- Differences in spacing, indentation, and comments
- Choice of the names given to variables (metavariables)
- Irrelevant code (‘. . .’, control flow oriented)
- Other variations in coding style (isomorphisms)

  e.g. if(!y) \equiv if(y==NULL) \equiv if(NULL==y)
Bug finding and fixing

- The "!&" bug

C allows mixing booleans and bit constants

```c
if (!state->card->ac97_status & CENTER_LFE_ON)
    val &= ~DSP_BIND_CENTER_LFE;
```

In `sound/oss/ali5455.c` until Linux 2.6.18

(problem is over two lines)
A Simple SmPL Sample

@@
expression E;
constant C;
@@

- !E & C   // !C is not a constant
+!(E & C)

- 96 instances in Linux
  from 2.6.13 (August 2005) to v2.6.28 (December 2008)
- 58 in 2.6.20 (February 2007),
- 2 in Linux-next (26th May 2009 and last Saturday)
Collateral Evolutions

Evolution becomes

Legend: before after

Collateral Evolutions (CE) in clients

lib.c

int foo(int x) {
int bar(int x, int y) {

client1.c

foo(1);
bar(1,?);
foo(2);
bar(2,?);

client2.c

foo(foo(2));
bar(bar(2,?),?);

clientn.c

if(foo(3)) {
if(bar(3,?)) {

}
Many libraries and many clients:

- Lots of driver support libraries: one per device type, one per bus (pci library, sound library, ...)
- Lots of device specific code: Drivers make up more than 50% of Linux

Many evolutions and collateral evolutions

1200 evolutions in 2.6, some affecting 400 files, at over 1000 sites [EuroSys 2006] (summer 2005)

Taxonomy of evolutions:

Add argument, split data structure, getter and setter introduction, protocol change, change return type, add error checking, ...
Example from Linux 2.5.71

- Evolution: `scsi_get()/scsi_put()` dropped from SCSI library
- Collateral evolutions: SCSI resource now passed directly to `proc_info` callback functions via a new parameter

```c
int a_proc_info(int x, scsi *y) {
    scsi *y;
    ...
    y = scsi_get();
    if(!y) { ... return -1; }
    ...
    scsi_put(y);  
    ...
}
```
Semantic Patches

```c
function a_proc_info;

identifier x, y;

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

Control-flow ‘...’ operator
Affected Linux driver code

drivers/scsi/53c700.c

```c
int s53c700_info(int limit)
{
    char *buf;
    scsi *sc;
    sc = scsi_get();
    if (!sc) {
        printk("error");
        return -1;
    }
    wd7000_setup(sc);
    PRINTP("val=%d",
        sc->field+limit);
    scsi_put(sc);
    return 0;
}
```

drivers/scsi/pcmcia/nsp_cs.c

```c
int nsp_proc_info(int lim)
{
    scsi *host;
    host = scsi_get();
    if (!host) {
        printk("nsp_error");
        return -1;
    }
    SPRINF("NINJASCSI=%d",
        host->base);
    scsi_put(host);
    return 0;
}
```

Similar, but not identical
Applying the semantic patch

```c
int s53c700_info(int limit)
{
    char *buf;
    scsi *sc;
    sc = scsi_get();
    if(!sc) {
        printk("error");
        return -1;
    }
    wd7000_setup(sc);
    PRINTP("val=%d",
            sc->field+limit);
    scsi_put(sc);
    return 0;
}
```

```c
int nsp_proc_info(int lim)
{
    scsi *host;
    host = scsi_get();
    if(!host) {
        printk("nsp_error");
        return -1;
    }
    SPRINTF("NINJASCSI=%d",
            host->base);
    scsi_put(host);
    return 0;
}
```

```c
int s53c700_info(int limit)
{
    char *buf;
    scsi *sc;
    sc = scsi_get();
    if(!sc) {
        printk("error");
        return -1;
    }
    wd7000_setup(sc);
    PRINTP("val=%d",
            sc->field+limit);
    scsi_put(sc);
    return 0;
}
```

```c
int nsp_proc_info(int lim)
{
    scsi *host;
    host = scsi_get();
    if(!host) {
        printk("nsp_error");
        return -1;
    }
    SPRINTF("NINJASCSI=%d",
            host->base);
    scsi_put(host);
    return 0;
}
```

$ spatch -sp_file proc_info.sp -dir linux-next

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

proc_info.sp
Applying the semantic patch

```
int s53c700_info(int limit, scsi *sc)
{
    char *buf;

    wd7000_setup(sc);
    PRINTP("val=%d", sc->field+limit);
    return 0;
}
```

```
int ns53c700_info(int limit, scsi *sc)
{
    char *buf;

    wd7000_setup(sc);
    PRINTP("val=%d", sc->field+limit);
    return 0;
}
```

```
int nsp_proc_info(int limit, scsi *host)
{
    SPRINTF("NINJASCSI=%d", host->base);
    return 0;
}
```

```
$ spatch -sp_file proc_info.sp -dir linux-next
```

```
@@
function a_proc_info;
identifier x,y;
@@
int a_proc_info(int x
+   ,scsi *y
   ) {
-   scsi *y;
-   ... 
-   y = scsi_get();
-   if(!y) { ... return -1; } 
-   ... 
-   scsi_put(y);
   ... 
}
```
Advance examples
fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
if (!fh) {
    dprintk(1, KERN_ERR
            "%s: zoran_open(): allocation of zoran_fh failed\n",
            ZR_DEVNAME(zr));
    return -ENOMEM;
}
memset(fh, 0, sizeof(struct zoran_fh));
fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
if (!fh) {
    dprintk(1, KERN_ERR "%s: zoran_open(): allocation of zoran_fh failed
" ZR_DEVNAME(zr));
    return -ENOMEM;
}
memset(fh, 0, sizeof(struct zoran_fh));

fh = kzalloc(sizeof(struct zoran_fh), GFP_KERNEL);
if (!fh) {
    dprintk(1, KERN_ERR "%s: zoran_open(): allocation of zoran_fh failed\n" ZR_DEVNAME(zr));
    return -ENOMEM;
}
Evolution:

kmalloc/memset ⇒ kzalloc

1) Eliminate irrelevant code

fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
...
memset(fh, 0, sizeof(struct zoran_fh));
Evolution:
kmalloc/memset ⇒ kzalloc

2) Describe the transformation

```c
-fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
+fh = kzalloc(sizeof(struct zoran_fh), GFP_KERNEL);
...
-memset(fh, 0, sizeof(struct zoran_fh));
```
Evolution:
kmalloc/memset ⇒ kzalloc

3) Abstract over subterms

```c
@@
expression x;
expression E1,E2;

@@
-x = kmalloc(E1, E2);
+x = kzalloc(E1, E2);
...

-memset(x, 0, E1);
```
4) Refinement

.Restricted

.expression x;
.expression E1, E2, E3;
.statement S;
.identifier f;

- \( x = \text{kmalloc}(E1, E2); \)
+ \( x = \text{kzalloc}(E1, E2); \)

... when \(!= ( f(...,x,...) | <+...x...+> = E3 )\)
    when \(!= ( \text{while (...)} S | \text{for (...;...;...)} S )\)
- \( \text{memset}(x, 0, E1); \)
Evolution: kmalloc/memset ⇒ kzalloc

5) Generalization

```c
@@
expression x;
expression E1, E2, E3;
statement S;
identifier f;
type T1, T2;
@@
-x = (T1)kmalloc(E1, E2);
+x = kzalloc(E1, E2);
... when != ( f(...,x,...) | <+...x...+> = E3 )
    when != ( while (...) S | for (...;...;...) S )
-memset((T2)x, 0, E1);
```

Updates 355/564 files
Evaluation on Collateral Evolutions
[Eurosys 2008]
Experiments

- Methodology
  - Detect past collateral evolutions in Linux 2.5 and 2.6 using the patchparse tool [Eurosys'06]
  - Select representative ones
    - Test suite of over 60 CEs
  - Study them and write corresponding semantic patches
    - Note: we are not kernel developers

- Going "back to the future". Compare:
  - What Linux programers did manually
  - What Coccinelle, given our SPs, does automatically
Test suite

- **20 Complex CEs**: bugs introduced by the programmers
  - In each case 1-16 errors + misses

- **23 Mega CEs**: affect over 100 sites on Linux between 2.6.12 and 2.6.20
  - 22-1124 files affected
  - Up to 39 human errors
  - Up to 40 people for up to two years

- **26 CEs for the bluetooth directory update from 2.6.12 to 2.6.20**
  - Median case

More than 5800 driver files
Results

- SP are on average 106 lines long (6-369)
- SPs often 100 times smaller than “human-made” patches. A measure of time saved:
  - Not doing manually the CE on all the drivers
  - Not reading and reviewing big patches, for people with drivers outside source tree
- Correct and complete automated evolutions for 93% of the files
  - Problems on the remaining 7%: We miss code sites
    - CPP issues, lack of isomorphisms (data-flow and inter-procedural)
    - We are not kernel developers ... don’t know how to specify
- Average processing time of 0.7s per file

Sometimes the tool was right and the human wrong
Impact on the Linux kernel

- Collateral evolution related SPs
  - Over 60 semantic patches

- SPs for bug-fixing and bad programming practices
  - Over 57 semantic patches

- Generated patches
  - Over 230 patches accepted
How does the Coccinelle tool work?
Transformation engine

Parse C file

Translate to CFG

Parse Semantic Patch

Expand isomorphisms

Translate to CTL

Match CTL against CFG using a model checking algorithm

Modify matched code

Unparse

Computational Tree Logic with extra features

[POPL 2009]
Other issues

- Need to produce readable code
  - Keep space, indentation, comments
  - Keep CPP instructions as-is. Also programmer may want to transform some `#define`, `iterator` macros (e.g. `list_for_each`)

Very different from most other C tools

- Interactive engine, partial match
- Implementation of isomorphisms
  - Rewriting the Semantic patch (not the C code),

68 000 lines of O'Caml code
Current/Future Work

Coccinelle in the large

- Semantic patch inference (spdiff) [ASE2008]
- Protocol-based bug detection in Linux [DSN2009]
- Enforcing API usage [ACP4IS2009]
- Herodotos: To study bugs’ lives [INRIA RR6984, CFSE2009]
- Collaborative design of rules
- Version consistency
Conclusion

- SmPL: a declarative language for program matching and transformation
- Quite “easy” to learn; already accepted by the Linux community
- SPs looks like a patch; fits with Linux programmers’ habits
- SPs documents evolutions
- A transformation engine based on model checking technology
Kill bugs before they hatch!!!

http://coccinelle.lip6.fr/