Coccinelle Features

Julia Lawall (Inria/LIP6)

http://coccinelle.lip6.fr

October 31, 2013

Coccinelle features

- Isomorphisms.
- Depends on.
- Dots.
- Positions.
- Python.

Isomorphisms

Issue:

- Coccinelle matches code exactly as it appears.
- `x == NULL` does not match `!x`.

Goal:

- Transparently treat similar code patterns in a similar way.

Example: DIV_ROUND_UP

The following code is fairly hard to understand:

```
return (time_ns * 1000 + tick_ps - 1) / tick_ps;
```

kernel.h provides the following macro:

```
#define DIV_ROUND_UP(n,d) (((n) + (d) - 1) / (d))
```

This is used, but not everywhere it could be.

We can write a semantic patch to introduce new uses.

DIV_ROUND_UP semantic patch

One option:

```
@@ expression n,d; @@

- (((n) + (d) - 1) / (d))
+ DIV_ROUND_UP(n,d)
```

Another option:

```
@@ expression n,d; @@

- (n + d - 1) / d
+ DIV_ROUND_UP(n,d)
```

Problem: How many parentheses to put, to capture all occurrences?

Isomorphisms

An isomorphism relates code patterns that are considered to be similar:

```
Expression
@ drop_cast @ expression E; pure type T; @@

 (T)E => E

Expression
@ paren @ expression E; @@

 (E) => E

Expression
@ is_null @ expression X; @@

 X == NULL <=> NULL == X => !X
```

Isomorphisms, contd.

Isomorphisms are handled by rewriting.

```
(((n) + (d) - 1) / (d))
```

becomes:

```
(
 (((n) + (d) - 1) / (d))
|
 (((n) + (d) - 1) / d)
|
 (((n) + d - 1) / (d))
|
 (((n) + d - 1) / d)
|
 ((n + (d) - 1) / (d))
|
 ((n + (d) - 1) / d)
|
 ((n + d - 1) / (d))
|
 ((n + d - 1) / d)
|
 etc.
)
```

Results

```
@@
expression n,d;
@@

- (((n) + (d) - 1) / (d))
+ DIV_ROUND_UP(n,d)
```

Changes 281 occurrences in Linux 3.2.

Practical issues

Default isomorphisms are defined in standard.iso

To use a different set of default isomorphisms:

```
spatch --sp-file mysp.cocci --dir linux-x.y.z --iso-file empty.iso
```

To drop specific isomorpshisms:

```
@disable paren@ expression n,d; @@
- (((n) + (d) - 1) / (d))
+ DIV_ROUND_UP(n,d)
```

To add rule-specific isomorphisms:

```
@using "myparen.iso" disable paren@
expression n,d;
@@
- (((n) + (d) - 1) / (d))
+ DIV_ROUND_UP(n,d)
```

Exercise 6

Some Linux code combines an assignment with a test, as illustrated by the following:

```
if (!(p = kmalloc(sz, GFP_KERNEL)))
  break;
```

The following semantic patch moves the assignment out of the conditional:

```
@@ identifier e1; expression e2; statement S1, S2; @@
+ e1 = e2;
  if (
-     (e1 = e2)
+     e1
      == NULL) S1 else S2
```

1. Test this semantic patch on linux-3.2/sound/pci/au88x0
2. How were isomorphisms used in these matches?

Exercise 7

Run

```
spatch --parse-cocci sp.cocci
```

For some semantic patch `sp.cocci` that you have developed.

Explain the result.

Depends on

```
@@
expression n,d;
@@

- (((n) + (d) - 1) / (d))
+ DIV_ROUND_UP(n,d)
```

Issue:

- `DIV_ROUND_UP` is a macro, defined in kernel.h.
- Maybe some file does not include kernel.h?
- `#include`, if present, would not be in the same function.

Depends on, contd.

```
@r@
@@
#include <linux/kernel.h>

@depends on r@
expression n,d;
@@
- (((n) + (d) - 1) / (d))
+ DIV_ROUND_UP(n,d)
```

Results:

- Naming a rule lets it be referenced by other rules.
- Only introduce `DIV_ROUND_UP` if the `#include` rule is satisfied.
- Matches 86 occurrences.

Dots

Issue:

- Sometimes it is necessary to search for multiple related code fragments.

Goals:

- Specify patterns consisting of fragments of code separated by arbitrary execution paths.
- Specify constraints on the contents of those execution paths.

Example: Inadequate error checking of kmalloc

`kmalloc` returns NULL on insufficient memory.

Good code:

```
block = kmalloc(WL12XX_HW_BLOCK_SIZE, GFP_KERNEL);
if (!block)
  return;
```

Bad code:

```
g = kmalloc (sizeof (*g), GFP_KERNEL);
g->next = chains[r_sym].next;
```

More bad code

```
alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
INIT_LIST_HEAD(&intmem_allocations);
intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                         MEM_INTMEM_SIZE - RESERVED_SIZE);
initiated = 1;
alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

The `kmalloc` and the dereference are not necessarily contiguous.

Using dots

Start with a typical example of code

```
alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
INIT_LIST_HEAD(&intmem_allocations);
intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                         MEM_INTMEM_SIZE - RESERVED_SIZE);
initiated = 1;
alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

Using dots

Highlight what is wanted

```
* alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
  INIT_LIST_HEAD(&intmem_allocations);
  intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                           MEM_INTMEM_SIZE - RESERVED_SIZE);
  initiated = 1;
* alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

Using dots

Replace the irrelevant statements by ...

```
* alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
  ...
```

```
* alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

Using dots

Abstract over irrelevant subterms.

- May use `...`.

```
@@ expression e; identifier f; @@
* e      = kmalloc(...);
  ...



* e->f
```

Using dots

Check properties of the matched statement sequence

```
@@ expression e; identifier f; @@
* e      = kmalloc(...);
  ... when != e == NULL
      when != e != NULL


* e->f
```

Using dots

Sanity check

```
@@ expression e, e1; identifier f; @@
* e      = kmalloc(...);
  ... when != e == NULL
      when != e != NULL
      when != e = e1

* e->f
```

Results: 18 kmallocs in 12 files

Real bug: linux-3.2/arch/cris/arch-v32/mm/intmem.c

```
- alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
  INIT_LIST_HEAD(&intmem_allocations);
  intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                           MEM_INTMEM_SIZE - RESERVED_SIZE);
  initiated = 1;
- alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

Results: 18 kmallocs in 12 files

Real bug: linux-3.2/arch/cris/arch-v32/mm/intmem.c

```
- alloc = kmalloc(sizeof *alloc, GFP_KERNEL);
  INIT_LIST_HEAD(&intmem_allocations);
  intmem_virtual = ioremap(MEM_INTMEM_START + RESERVED_SIZE,
                           MEM_INTMEM_SIZE - RESERVED_SIZE);
  initiated = 1;
- alloc->size = MEM_INTMEM_SIZE - RESERVED_SIZE;
```

False positive! linux-3.2/net/ipv4/syncookies.c

```
- ireq->opt = kmalloc(opt_size, GFP_ATOMIC);
- if (ireq->opt != NULL && ip_options_echo(&ireq->opt->opt, skb)) {
    kfree(ireq->opt);
    ireq->opt = NULL;
  }
```

False positives

```
ireq->opt != NULL && ip_options_echo(&ireq->opt->opt, skb)
```

"`...`" matches complete statements.

- `ireq->opt != NULL` is not seen as being before `&ireq->opt->opt`.

Solution: stop at NULL tests or bad dereference (disjunction).

- `e == NULL`: OK
- `e != NULL`: OK
- `e->f`: Bug

Revised version

```
@@ expression e,e1; identifier f; @@
  e = kmalloc(...);
  ... when != e = e1
(
  e == NULL || ...
|
  e != NULL && ...
|
* e->f
)
```

Shortest path property:

- "..." matches everything except what is on either side.

Matches 11 files, eliminating the false positive.

Exercise 8

The following code allocates a region of memory and then clears it:

```
state = kmalloc(sizeof(struct drxd_state), GFP_KERNEL);
if (!state)
  return NULL;
memset(state, 0, sizeof(*state));
```

The function `kzalloc` does both, *i.e.*, we could write:

```
state = kzalloc(sizeof(struct drxd_state), GFP_KERNEL);
if (!state)
  return NULL;
```

1. Write a semantic patch to make this transformation.
2. Test your semantic patch on `linux-3.2/drivers/net/wireless`.
3. Are there any files where your semantic patch should not transform the code, but it does?

Exercise 9

One of the results for the `kmalloc` with no NULL test example is the following (linux-3.2/drivers/macintosh/via-pmu.c):

```
- pp = kmalloc(sizeof(struct pmu_private), GFP_KERNEL);
  if (pp == 0)
     return -ENOMEM;
- pp->rb_get = pp->rb_put = 0;
```

The code will not crash, but it is not as nice as it could be. Write a semantic patch to replace such bad uses of `0` by `NULL`.

Hints:

- A metavariable of type "`expression *`" matches any pointer typed expression.
- This exercise has nothing to do with dots.

Positions and Python

```
@@ expression e,e1; identifier f; @@
  e = kmalloc(...);
  ... when != e = e1
(
  e == NULL || ...
|
  e != NULL && ...
|
* e->f
)
```

Output reported as a diff:

- Useful in emacs (diff-mode).
- Perhaps less useful in other contexts.

Bonus question: Why is there no * on `kmalloc`?

Positions and Python

Goal:

- Collect positions of some matched elements.
- Print a helpful error message.

```
@r@
expression e,e1;
identifier f;
position p1, p2;
@@
  e = kmalloc@p1(...);
  ... when != e = e1
(
  e == NULL || ...
|
  e != NULL && ...
|
  e@p2->f
)
```

```
@script:python@
p1 << r.p1;
p2 << r.p2;
@@
l1 = p1[0].line
l2 = p2[0].line
print "kmalloc on line %s not tested
       before reference on line %s" %
       (l1,l2)
```

A refinement

Exists:

- Require only a single matching execution path.
- Default for *.

```
@r exists@
expression e,e1;
identifier f;
position p1, p2;
@@
  e = kmalloc@p1(...);
  ... when != e = e1
(
  e == NULL || ...
|
  e != NULL && ...
|
  e@p2->f
)
```

```
@script:python@
p1 << r.p1;
p2 << r.p2;
@@
l1 = p1[0].line
l2 = p2[0].line
print "kmalloc on line %s not tested
       before reference on line %s" %
       (l1,l2)
```

Exercise 10

Rewrite a semantic patch that you have implemented previously, so that it prints the line numbers on which a change is needed, rather than making the change.

Useful terms:

- `p[0].file` is the name of the file represented by `p`.
- `p[0].line` is the number, as a string, of the line represented by `p`.
- `p` is an array, because there can be many matches.

Summary

- Isomorphisms, for simplifying, eg NULL tests, parentheses, casts.
- Dots, for matching a sequence of statements, arguments, etc.
- When, for restricting the contents of sequences.
- Positions, for remembering the exact position of some code.
- Python, for printing error messages, managing hashtables, etc.