Coccinelle

Julia Lawall (Inria/Irill/LIP6)

http://coccinelle.lip6.fr
http://btrlinux.inria.fr

August 19, 2015

Goal

Help developers scan and transform a large legacy C code base.

Applications:

- Bug finding
  - A developer finds a bug in one part of the code and wants to see if it occurs elsewhere.
- Bug fixing
  - Modifying the code by hand can leave it in worse shape than it started out.
- Code modernization
  - Improved API functions are often introduced, but not always pervasively used.
- Code metrics
  - How many times is function XXX used outside a probe function?

Coccinelle

Find once, fix everywhere.

Approach:

- Static analysis to find patterns in C code.
- Automatic transformation to fix bugs.
- User scriptable, based on patch notation (semantic patches).
- http://coccinelle.lip6.fr/, http://btrlinux.inria.fr/

Coccinelle

Find once, fix everywhere.

Approach:

- Static analysis to find patterns in C code.
- Automatic transformation to fix bugs.
- User scriptable, based on patch notation (semantic patches).
- `http://coccinelle.lip6.fr/`, `http://btrlinux.inria.fr/`

Goal: Be accessible to C code developers.

Bug: !x&y

```
Author: Al Viro <viro@ZenIV.linux.org.uk>

    wmi: (!x & y) strikes again

diff --git a/drivers/acpi/wmi.c b/drivers/acpi/wmi.c
@@ -247,7 +247,7 @@
  block = &wblock->gblock;
  handle = wblock->handle;

- if (!block->flags & ACPI_WMI_METHOD)
+ if (!(block->flags & ACPI_WMI_METHOD))
    return AE_BAD_DATA;

  if (block->instance_count < instance)
```

How to automate this change?

- For any `!E & C`
  - where `E` is any expression, and
  - where `C` is any constant,
- Add parentheses around `E & C`

Finding and fixing !x&y bugs using Coccinelle

```
@@
expression E;
constant C;
@@

- !E & C
+ !(E & C)
```

- E is an arbitrary expression.
- C is an arbitrary constant.

Example

Original code:

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

Semantic patch:

```
@@ expression E; constant C; @@
- !E & C
+ !(E & C)
```

Generated code:

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

API-specific issues

Dangerous code:

```
static int wp384_final(struct shash_desc *desc, u8 *out)
{
        u8 D[64];

        wp512_final(desc, D);
        memcpy (out, D, WP384_DIGEST_SIZE);
        memset (D, 0, WP512_DIGEST_SIZE);

        return 0;
}
```

API-specific issues

Dangerous code:

```
static int wp384_final(struct shash_desc *desc, u8 *out)
{
        u8 D[64];

        wp512_final(desc, D);
        memcpy (out, D, WP384_DIGEST_SIZE);
        memset (D, 0, WP512_DIGEST_SIZE);

        return 0;
}
```

The compiler can optimize away the call to `memset`.

Solution

```
void memzero_explicit(void *s, size_t count)
{
        memset(s, 0, count);
        OPTIMIZER_HIDE_VAR(s);
}
```

Want to use this if and only if needed.

Automating the introduction of memzero_explicit

```
- memset(x,0,ct)
+ memzero_explicit(x,ct)
```

Automating the introduction of memzero_explicit

```
@@
identifier x;
expression ct;

@@




- memset(x,0,ct)
+ memzero_explicit(x,ct)
```

Automating the introduction of memzero_explicit

```
@@
identifier x;
expression ct;

@@




- memset(x,0,ct)
+ memzero_explicit(x,ct)
  ... when != x
```

Automating the introduction of memzero_explicit

```
@@
identifier x;
expression ct;
type T;
@@

  T x[...];
  ... when any

- memset(x,0,ct)
+ memzero_explicit(x,ct)
  ... when != x
```

Automating the introduction of memzero_explicit

```
@@
identifier x;
expression ct;
type T;
@@

  T x[...];
  ... when any
      when exists
- memset(x,0,ct)
+ memzero_explicit(x,ct)
  ... when != x
```

A good result

```
static int wp384_final(struct shash_desc *desc, u8 *out)
{
        u8 D[64];

        wp512_final(desc, D);
        memcpy (out, D, WP384_DIGEST_SIZE);
-       memset (D, 0, WP512_DIGEST_SIZE);
+       memzero_explicit(D, WP512_DIGEST_SIZE);
        return 0;
}
```

A false positive

```
struct mgmt_ev_device_found *ev;
char buf[sizeof(*ev) + HCI_MAX_NAME_LENGTH + 2];
u16 eir_len;

ev = (struct mgmt_ev_device_found *) buf;
memset(buf, 0, sizeof(buf));
bacpy(&ev->addr.bdaddr, bdaddr);
ev->addr.type = link_to_bdaddr(link_type, addr_type);
ev->rssi = rssi;
```

A false positive

```
struct mgmt_ev_device_found *ev;
char buf[sizeof(*ev) + HCI_MAX_NAME_LENGTH + 2];
u16 eir_len;

ev = (struct mgmt_ev_device_found *) buf;
memset(buf, 0, sizeof(buf));
bacpy(&ev->addr.bdaddr, bdaddr);
ev->addr.type = link_to_bdaddr(link_type, addr_type);
ev->rssi = rssi;
```

A false positive

```
struct mgmt_ev_device_found *ev;
char buf[sizeof(*ev) + HCI_MAX_NAME_LENGTH + 2];
u16 eir_len;

ev = (struct mgmt_ev_device_found *) buf;
memset(buf, 0, sizeof(buf));
bacpy(&ev->addr.bdaddr, bdaddr);
ev->addr.type = link_to_bdaddr(link_type, addr_type);
ev->rssi = rssi;
```

Refining the semantic patch

```
@@
identifier x;
type T,T1;
expression e,ct;
@@

T x[...];
... when any
    when exists
    when != e = (T1)x
    when != e = (T1)&x[0]
- memset(x,0,ct)
+ memzero_explicit(x,ct)
... when != x
```

Refining the semantic patch

```
@@
identifier x;
type T,T1;
expression e,ct;
@@

T x[...];
... when any
    when exists
    when != e = (T1)x
    when != e = (T1)&x[0]
- memset(x,0,ct)
+ memzero_explicit(x,ct)
... when != x
```

Around 30 uses in recent versions of Linux

A new feature: Coccinelle 1.0.2

Obtaining the statement that contains an expression:

```
@r@
expression e1, e2; identifier f; position p; statement S;
@@
f(...,e1 = e2,...)@S@p

@@
expression r.e1, r.e2; statement S; position r.p;
@@
++ e1=e2;
   S@p

@@
expression r.e1,r.e2; identifier r.f;
@@
  f(...,
  e1
- = e2
  ,...)
```

Some examples

drivers/gpu/drm/i915/intel_lrc.c:

```
- return wa_ctx_end(wa_ctx, *offset = index, 1);
+ *offset = index;
+ return wa_ctx_end(wa_ctx, *offset, 1);
```

drivers/ide/qd65xx.c:

```
- if (timings[index] != QD_TIMING(drive))
-   outb(timings[index] = QD_TIMING(drive), QD_TIMREG(drive));
+ if (timings[index] != QD_TIMING(drive)) {
+   timings[index] = QD_TIMING(drive);
+   outb(timings[index], QD_TIMREG(drive));
+ }
```

A future feature?

Matching simultaneous patterns:

```
@r@
expression e1,e2;
identifier f;
position p;
statement S;
@@

(
++  e1=e2;
    S@p
&
    f(...,e1
-            = e2
      ,...)@S@p
)
```

Conclusion

Coccinelle:

- Code-like matching and transformation language.
- Flexibility via a small set of features (dots, disjunction, etc.)
- Interface with python and ocaml.
- False positives possible, but can be controlled, by adjusting the rules or manual intervention.

Status:

- 3283 Linux kernel patches mention Coccinelle
- 53 semantic patches in the Linux kernel source tree
- Many examples at coccinellery.org
- Other resources at coccinelle.lip6.fr, btrlinux.inria.fr