Coccinelle: Finding bugs in the Linux Kernel

FOSSASIA 2017, Singapore

Vaishali Thakkar

(vaishali.thakkar@oracle.com, @kernel_girl)

Who Am I?

- Linux Kernel developer at Oracle
- Working in kernel security engineering group
- Interested in many different subsystems of the Linux Kernel
- Associated with the open source internship programs

Agenda

- Need of Coccinelle
- Introduction to Coccinelle with examples
- Spatch and coccicheck
- Some notable other features of Coccinelle

Code Maintenance Issues

- **Software evolution:**
  - Refactoring code to use newer APIs
  - Need to find all parts of the code that need updating
  - Process should be fast, reliable and systematic
  - However, things are never straightforward

Code Maintenance Issues

- **Software evolution:**
  - Refactoring code to use newer APIs
  - Need to find all parts of the code that need updating
  - Process should be fast, reliable and systematic
  - However, things are never straightforward
- **Software robustness:**
  - Are the programmers following the standards?
  - Is the code accounting for all errors that can take place?

Code Maintenance Issues

- **Software evolution:**
  - Refactoring code to use newer APIs
  - Need to find all parts of the code that need updating
  - Process should be fast, reliable and systematic
  - However, things are never straightforward
- **Software robustness:**
  - Are the programmers following the standards?
  - Is the code accounting for all errors that can take place?
- **The Human Factor:**
  - Copy-pasting the code and mistakes can always happen

Example 1: Using BIT macro

- Bit masking is preferrably done using the BIT macro

```
-       BUILD_BUG_ON(max >= (1 << 16));
+       BUILD_BUG_ON(max >= (BIT(16)));
```

Example 2: Some useless code

```
diff --git a/drivers/staging/dgnc/dgnc_tty.c b/drivers/staging/dgnc/dgnc_t
index 0e903dc..a4e6c9e 100644
--- a/drivers/staging/dgnc/dgnc_tty.c
+++ b/drivers/staging/dgnc/dgnc_tty.c
@@ -2286,7 +2286,6 @@ static inline int dgnc_get_mstat(struct channel_t *c
 static int dgnc_get_modem_info(struct channel_t *ch, unsigned int  __user
 {
        int result;
-       int rc;

        if (!ch || ch->magic != DGNC_CHANNEL_MAGIC)
                return -ENXIO;
@@ -2296,9 +2295,7 @@ static int dgnc_get_modem_info(struct channel_t *ch,
        if (result < 0)
                return -ENXIO;

-       rc = put_user(result, value);
-
-       return rc;
+       return put_user(result, value);
 }
```

Example 3: Some serious stuff

```
static long hwdep_read_locked( ...)
{
        ...
        if (copy_to_user(buf, &event, count))
                return -EFAULT;
        ...
}

static long hwdep_read( ...)
{
        ...
        while (...) {
                spin_lock_irq ( ...);
        }

        if (efw->dev_lock_changed)
                count = hwdep_read_locked( ...);
        ...
        spin_unlock_irq ( ...);
        ...
}
```

Example 3: Some serious stuff

```
static long hwdep_read_locked(...)
{
        ...
        if (copy_to_user(buf, &event, count))
                return -EFAULT;
        ...
}

static long hwdep_read(...)
{
        ...
        while (...) {
                spin_lock_irq(...);
        }

        if (efw->dev_lock_changed)
                count = hwdep_read_locked(...);
        ...
        spin_unlock_irq(...);
        ...
}
```

Goals of the tool Coccinelle

- Automatically find the bugs, based on the pattern
  - Ability to abstract over irrelevant information
  - Ability to transform over scattered code fragements
- Automatically fix the bugs, based on the pattern
  - Ability to transform the code fragements
- Providing a system accessible to the software developers
- Providing a system separate from architectures and compilation process

Coccinelle

- Program matching and transformation tool
- Have it's own language called Semantic patch Language [SmPL]
- Very intuitive patch like notation style
- Used by several communities:
  - Linux Kernel: 5K+ patches
  - QEMU: 200+ patches
  - systemd: 80+ patches

Semantic Patch Language (SmPL)

- Abstract C-like grammar
- Independent of the compilation process
- Metavariables: Used to abstract over sub-terms in code
  - If an expression matches within a pattern, it can be tracked throughout its presence in the code e.g. variable names, typedefs
- "..." is used to abstract over code sequences
  - Used as don't care
  - Variants are used as syntactic sugar for + and ? in regular expressions
- Lines can be annotated with {-,+,*}

Example 1: Using BIT macro

- Bit masking is preferrably done using the BIT macro

```
-       BUILD_BUG_ON(max >= (1 << 16));
+       BUILD_BUG_ON(max >= (BIT(16)));
```

Example 1: Using BIT macro

- Bit masking is preferrably done using the BIT macro

```
-       BUILD_BUG_ON(max >= (1 << 16));
+       BUILD_BUG_ON(max >= (BIT(16)));
```

- Code we should focus on for building a semantic patch:

```
- 1 << 16
+ BIT(16)
```

Example 1: Using BIT macro

- Bit masking is preferrably done using BIT macro

```
-       BUILD_BUG_ON(max >= (1 << 16));
+       BUILD_BUG_ON(max >= (BIT(16)));
```

- Code we should focus on for building a semantic patch:

```
- 1 << 16
+ BIT(16)
```

- Is 16 important here?

Example 1: Using BIT macro

- Do we care about number of shifts?

```
-       if (opts & (1 << REISERFS_LARGETAIL))
+       if (opts & (BIT(REISERFS_LARGETAIL)))
```

Example 1: Using BIT macro

- Do we care about number of shifts?

```
- if (opts & (1 << REISERFS_LARGETAIL))
+ if (opts & (BIT(REISERFS_LARGETAIL)))
```

- Use of metavariable:

```
@@
constant c;
@@

-1 << c
+BIT(c)
```

Example 1: Using BIT macro

- Constant will capture numbers and defined constants
- What if we had something like

```
1 << (31 - inode->i_sb->s_blocksize_bits)
```

Example: Using BIT macro

- Use of disjunction (semantic patch):

```
@@
constant c;
expression E;
@@

(
-1 << c
+BIT(c)
|
-1 << E
+BIT(E)
)
```

Example 1: Using BIT macro

- Would like to restrict the bitmask semantic patch to files that are already using the BIT macro?

```
diff -u -p a/arch/mips/pci/pci-mt7620.c b/arch/mips/pci/pci-mt7620.c
--- a/arch/mips/pci/pci-mt7620.c
+++ b/arch/mips/pci/pci-mt7620.c
@@ -37,11 +37,11 @@
 #define PDRV_SW_SET                     BIT(23)

 #define PPLL_DRV                        0xa0
-#define PDRV_SW_SET                     (1<<31)
-#define LC_CKDRVPD                      (1<<19)
-#define LC_CKDRVOHZ                     (1<<18)
-#define LC_CKDRVHZ                      (1<<17)
-#define LC_CKTEST                       (1<<16)
+#define PDRV_SW_SET                     (BIT(31))
+#define LC_CKDRVPD                      (BIT(19))
+#define LC_CKDRVOHZ                     (BIT(18))
+#define LC_CKDRVHZ                      (BIT(17))
+#define LC_CKTEST                       (BIT(16))
```

Example 1: Using BIT macro

- Using multiple rules (semantic patch):

```
@usesbit@
@@
BIT(...)

@depends on usesbit@
expression E;
@@

- 1 << E
+ BIT(E)
```

Example 2: Compressing the lines

- In the following code we don't care about the arguments of function snprintf:

```
int bytes_written;
u16 link_speed;

link_speed = rtw_get_cur_max_rate(padapter) / 10;
bytes_written = snprintf(command, total_len, "LinkSpeed %d", link_speed);
return bytes_written;
```

```
int bytes_written;
u16 link_speed;

link_speed = rtw_get_cur_max_rate(padapter) / 10;
return snprintf(command, total_len, "LinkSpeed %d", link_speed);
```

Example 2: Compressing the lines

Using Dots (semantic patch):

```
@@
expression r;
identifier f;
@@

-r = f(...)
+return
     f(...);
-return r;
```

Transformation specification

- - in the leftmost column for something to remove
- + in the leftmost column for something to add
- * in the leftmost column for something of interest
  - Cannot be used with + and -.
- Spaces, newlines that are irrelevant.

Spatch

- Coccinelle's command-line tool
- To check that your semantic patch is valid:

```
spatch --parse-cocci mysp.cocci
```

- To run your semantic patch:

```
spatch --sp-file mysp.cocci file.c
```

```
spatch --sp-file mysp.cocci --dir directory
```

Coccicheck

- A Coccinelle-specific target which is defined in the top level Makefile.
- Four basic modes
  - Patch mode
  - Context mode
  - Org mode
  - Report mode
- Default output: Report mode
- Command: make coccicheck MODE=patch

Modes for the Coccinelle script

- Four basic modes
  - Patch mode: proposes a fix when possible.

```
@@ -582,8 +580,7 @@ static int iss_net_configure(int index,
         return 1;
 }

-        init_timer(&lp->tl);
-        lp->tl.function = iss_net_user_timer_expire;
+        setup_timer(&lp->tl, iss_net_user_timer_expire, 0UL);

     return 0;
```

Modes for the Coccinelle script

- Four basic modes
  - Context mode:

1. highlights lines of interest and their context in a diff-like style.
2. Lines of interest are indicated with '-'.

```
@@ -582,8 +580,7 @@ static int iss_net_configure(int index,
return 1;
}

-       init_timer(&lp->tl);
-       lp->tl.function = iss_net_user_timer_expire;
-       setup_timer(&lp->tl, iss_net_user_timer_expire, 0UL);

    return 0;
```

Modes for the Coccinelle script

- Four basic modes
  - Org mode: Generates a report in the Org mode format of Emacs.

```
* TODO [[view:/arch/sh/drivers/pci/common.c::face=ovl-face1::linb=1
::cole=12] [Use setup_timer function.]]
[[view:arch/sh/drivers/pci/common.c::face=ovl-face1::linb=109::colb
[arch/sh/drivers/pci/common.c::109]]

* TODO [[view:arch/sh/drivers/pci/common.c::face=ovl-face1::linb=11
::cole=12] [Use setup_timer function.]]
[[view:arch/sh/drivers/pci/common.c::face=ovl-face1::linb=115::colb
[arch/sh/drivers/pci/common.c::115]]
```

Modes for the Coccinelle script

- Four basic modes
  - Report mode: Generates a list in the following format

file:line:column-column: message

```
arch/sh/drivers/pci/common.c:108:2-12: Use setup_timer function
around line 109.
arch/sh/drivers/pci/common.c:114:2-12: Use setup_timer function
around line 115.
arch/sh/drivers/push-switch.c:81:1-11: Use setup_timer function
around line 83.
arch/x86/kernel/pci-calgary_64.c:1010:1-11: Use setup_timer functic
around line 1011.
arch/powerpc/oprofile/op_model_cell.c:682:1-11: Use setup_timer
function around line 683.
```

Some other notable features

- Embedding Python/ocaml script
  - Python/ocaml code is used inside special SmPL rule annotated with script:python/script:ocaml.
- Isomorphisms: Pattern 'return ...;' can match 'return;'
- Putting constraints
  - Using when between two code fragements
- Position metavariables
  - Store the position of any token, for later matching/printing.
- Iteration `
  - Iterating coccinelle with ocaml/python scripting

Useful links

- Source code of the Coccinelle: "https://github.com/coccinelle/coccinelle"
- Grammar and features: "http://coccinelle.lip6.fr/docs/options.pdf"
- Project: "http://coccinelle.lip6.fr/"
- Coccicheck: "https://bottest.wiki.kernel.org/coccicheck"
- Spatch: "https://www.mankier.com/1/spatch"

Thank You!