Inside the Mind of a Coccinelle Programmer

Julia Lawall
(Inria/LIP6)

Linux Security Summit
August 25, 2016

What is Coccinelle?

Find once, fix everywhere.

Approach: Coccinelle: `http://coccinelle.lip6.fr/`

- Static analysis to find patterns in C code.
- Automatic transformation to perform evolutions and fix bugs.
- User scriptable, based on patch notation (semantic patches).

What is Coccinelle?

Find once, fix everywhere.

Approach: Coccinelle: `http://coccinelle.lip6.fr/`

- Static analysis to find patterns in C code.
- Automatic transformation to perform evolutions and fix bugs.
- User scriptable, based on patch notation (semantic patches).

Goal: Be accessible to C code developers.

A classical example

```
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)
```

Using Coccinelle

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

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

Use case

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:

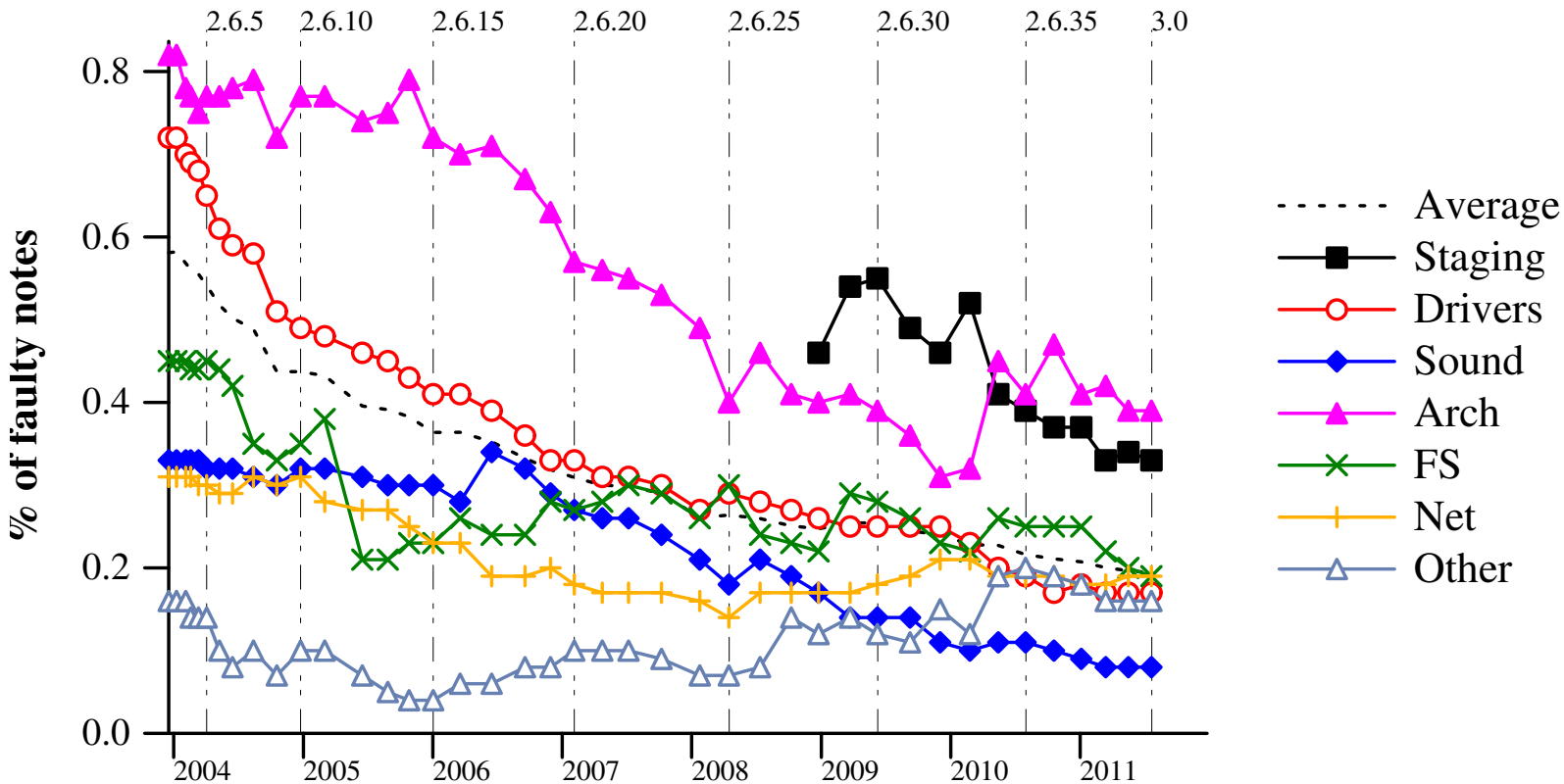
```
if (!(state->card->ac97_status & CENTER_LFE_ON))
        val &= ~DSP_BIND_CENTER_LFE;
```

Some history

- Initiated in 2004 while on sabbatical at EMN.
  - Identified the problem of "collateral evolution".
- Initially developed at the University of Copenhagen/EMN - 2005-2007
  - Julia Lawall - programming languages, now at Inria
  - Gilles Muller - systems, now at Inria
  - René Rydhof Hansen - security, now at U of Aalborg
  - Yoann Padioleau - postdoc, later at Facebook (pfff)
- First patches submitted to the Linux kernel in 2007
  - kmalloc + memset $\rightarrow$ kzalloc
- Language published at EuroSys in 2008.
- Faults in Linux published at ASPLOS in 2011.

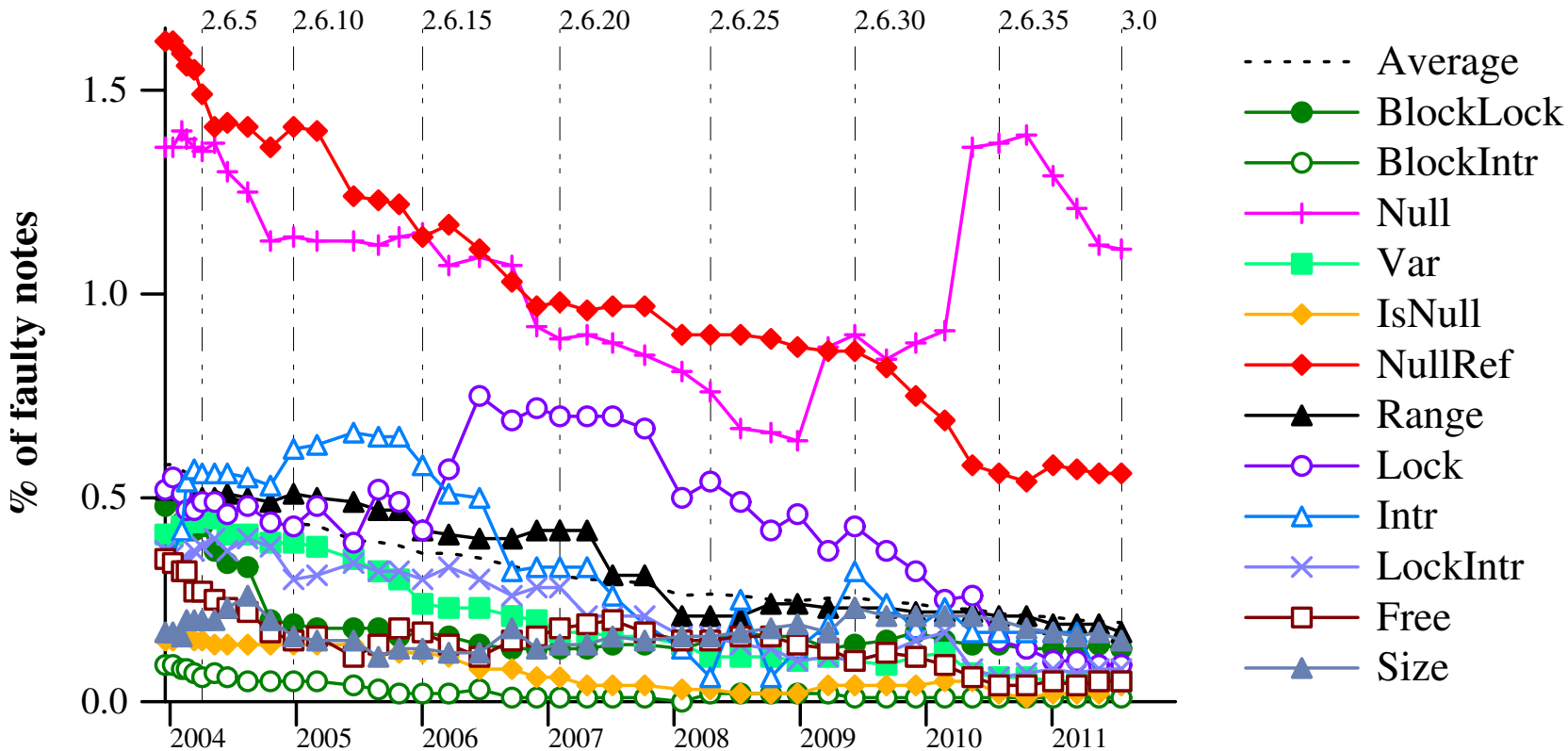
Faults in Linux

Fault rate



Faults in Linux

Fault kinds



Current status

- Developed at Inria.
  - Julia Lawall, Gilles Muller - fulltime researchers
  - Thierry Martinez - Inria engineer
  - Quentin Lambert - Inria young engineer (btrlinux.fr)
- Occasional external contributions.
- Entirely implemented in OCaml

Impact on the Linux kernel

- First patches submitted to the Linux kernel in 2007
  - kmalloc $\rightarrow$ memset
- Over 4500 patches in the Linux kernel mention Coccinelle
  - 3000 from over 500 developers outside our research group
- 56 semantic patches in the Linux kernel
  - Usable via make coccicheck
- Other Linux-related semantic patches at http://coccinellery.org/

Some semantic patches in the Linux kernel

- Generic C errors.
  - Unsigned less than 0.
- Generic Linux errors.
  - Double lock.
  - Use of iterator index after loop.
- API-specific Linux errors.
  - Free after `devm` allocation.
- API modernizations.
  - Use `kmemdup`, instead of `kmalloc` and `memcpy`.

Some semantic patches in the Linux kernel

- Generic C errors.
  - Unsigned less than 0.
- Generic Linux errors.
  - Double lock.
  - Use of iterator index after loop.
- API-specific Linux errors.
  - Free after `devm` allocation.
- API modernizations.
  - Use `kmemdup`, instead of `kmalloc` and `memcpy`.

Contributions welcome!

Some semantic patches in Linux kernel commits

Example:

```
@@
struct ethtool_ops *ops;
struct net_device *dev;
@@
-    SET_ETHTOOL_OPS(dev, ops);
+    dev->ethtool_ops = ops;
```

Example:

```
@@
struct gpio_chip *var;
@@
-var->dev
+var->parent
```

Some more complex applications

Overview

- Devmification: Manage driver memory to avoid memory leaks.
- Blocking functions under locks: potential deadlock.
- Constification: Protect function pointers in data structures.

Devmification

Example:

```
static int pxa3xx_u2d_probe(struct platform_device *pdev)
{
  int err;
  u2d = kzalloc(sizeof(struct pxa3xx_u2d_ulpi), GFP_KERNEL);
  if (!u2d) return -ENOMEM;
  u2d->clk = clk_get(&pdev->dev, NULL);
  if (IS_ERR(u2d->clk)) {
    err = PTR_ERR(u2d->clk);
    goto err_free_mem;
  }
  ...
  return 0;
err_free_mem:
  kfree(u2d);
  return err;
}
static int pxa3xx_u2d_remove(struct platform_device *pdev)
{
  clk_put(u2d->clk);
  kfree(u2d);
  return 0;
}
```

Devmification

Example:

```
static int pxa3xx_u2d_probe(struct platform_device *pdev)
{
  int err;
  u2d = kzalloc(sizeof(struct pxa3xx_u2d_ulpi), GFP_KERNEL);
  if (!u2d) return -ENOMEM;
  u2d->clk = clk_get(&pdev->dev, NULL);
  if (IS_ERR(u2d->clk)) {
    err = PTR_ERR(u2d->clk);
    goto err_free_mem;
  }
  ...
  return 0;
err_free_mem:
  kfree(u2d);
  return err;
}
static int pxa3xx_u2d_remove(struct platform_device *pdev)
{
  clk_put(u2d->clk);
  kfree(u2d);
  return 0;
}
```

Devmification

Example:

```
static int pxa3xx_u2d_probe(struct platform_device *pdev)
{
  int err;
  u2d = devm_kzalloc(&pdev->dev, sizeof(struct pxa3xx_u2d_ulpi), GFP_KERNEL);
  if (!u2d) return -ENOMEM;
  u2d->clk = clk_get(&pdev->dev, NULL);
  if (IS_ERR(u2d->clk)) {
    err = PTR_ERR(u2d->clk);
    goto err_free_mem;
  }
  ...
  return 0;
err_free_mem:

  return err;
}
static int pxa3xx_u2d_remove(struct platform_device *pdev)
{
  clk_put(u2d->clk);

  return 0;
}
```

Devmification: step by step

Find probe and remove functions from driver structure.

```
@platform@
identifier p, probefn, removefn;
@@
struct platform_driver p = {
  .probe = probefn,
  .remove = removefn,
};
```

Devmification: step by step

Update allocations in the probe function

```
@prb@
identifier platform.probefn, pdev;
expression e, e1, e2;
@@
probefn(struct platform_device *pdev, ...) {
  <+...
- e = kzalloc(e1, e2)
+ e = devm_kzalloc(&pdev->dev, e1, e2)


  ...+>
}
```

Devmification: step by step

Drop frees from probe...

```
@prb@
identifier platform.probefn, pdev;
expression e, e1, e2;
@@
probefn(struct platform_device *pdev, ...) {
  <+...
- e = kzalloc(e1, e2)
+ e = devm_kzalloc(&pdev->dev, e1, e2)
  ...
?-kfree(e);
  ...+>
}
```

Devmification: step by step

... and from remove

```
@rem depends on prb@
identifier platform.removefn;
expression prb.e;
@@
removefn(...) {
  <...
- kfree(e);
  ...>
}
```

Devmification: assessment

- 39 patches submitted, $\sim$ 2012.
- Opportunities in 171 files detected by the above semantic patch.
- Under 30 seconds wallclock time on an (old) 8-core Intel Xeon X5450 @ 3.00GHz, with glimpse indexing.

Locks on blocking functions

- Some functions that interact with user level block due to page faults.
  - `copy_from_user`, `copy_to_user`, `get_user`, `put_user`, etc.
- Blocking is not allowed under a spin lock
  - `spin_lock`, `spin_lock_irqsave`, `spin_lock_irq`, `spin_lock_bh`, `spin_trylock`, etc.
  - read and write variants

Goal: Detect user level calls that may be under a spin lock.

Locks on blocking functions

Intraprocedural case (spin locks only):

```
@@
@@
( spin_lock(...)
| spin_lock_irqsave(...)
| spin_lock_irq(...)
| spin_lock_bh(...)
| spin_trylock(...)
)
  ... when != spin_unlock(...)
      when != spin_unlock_irqrestore(...)
      when != spin_unlock_irq(...)
      when != spin_unlock_bh(...)
* copy_from_user(...)
```

Locks on blocking functions

Intraprocedural case (spin locks only):

```
@@
@@
( spin_lock(...)
| spin_lock_irqsave(...)
| spin_lock_irq(...)
| spin_lock_bh(...)
| spin_trylock(...)
)
  ... when != spin_unlock(...)
      when != spin_unlock_irqrestore(...)
      when != spin_unlock_irq(...)
      when != spin_unlock_bh(...)
* copy_from_user(...)
```

Using iteration, scales up to inter-procedural or inter-file analysis.

Locks on blocking functions: Results

`copy_from_user`:

- `drivers/gpu/drm/msm/msm_gem_submit.c` (confirmed bug)

`copy_to_user`:

- `drivers/usb/gadget/legacy/inode.c` (fixme)
- `drivers/usb/gadget/function/f_fs.c` (false positive)

`get_user`, `put_user`:

- 1 each in `arch/blackfin/kernel/sys_bfin.c`

Work in progress, with Vaishali Thakkar.

Constification

Example:

```
static struct dca_ops ioat_dca_ops = {
  .add_requester          = ioat_dca_add_requester,
  .remove_requester       = ioat_dca_remove_requester,
  .get_tag                = ioat_dca_get_tag,
  .dev_managed            = ioat_dca_dev_managed,
};
```

Constification

Example:

```
static const struct dca_ops ioat_dca_ops = {
  .add_requester          = ioat_dca_add_requester,
  .remove_requester       = ioat_dca_remove_requester,
  .get_tag                = ioat_dca_get_tag,
  .dev_managed            = ioat_dca_dev_managed,
};
```

Constification: step by step

Multi-step approach:

- Coccinelle: Find structures that only contain functions.
- Manually: Choose a promising candidate type.
- Coccinelle: Update all uses of the type with `const`.
- Manually: Compile test

Constification: semantic patch (step 2)

```
@r@
identifier virtual.ty;
position p;
@@
const struct ty@p

@@
identifier virtual.ty;
position p != r.p;
@@
+ const
struct ty@p
```

Results:

- 115 patches submitted in $\sim$ 2015.
- Detecting structures with only function fields is slow, constifying one structure is immediate.

Lessons learned

Start simple

Start with a semantic patch that matches the common case.

- Check other cases by hand, or
- Automate when more expertise is acquired.

Start simple

Start with a semantic patch that matches the common case.

- Check other cases by hand, or
- Automate when more expertise is acquired.

Devmification:

- Want to remove `kfree` from the probe and remove functions.
- `kfree` of what value?

Start simple

Start with a semantic patch that matches the common case.

- Check other cases by hand, or
- Automate when more expertise is acquired.

Devmification:

- Want to remove `kfree` from the probe and remove functions.
- `kfree` of what value?
- Simple solution: assume the value always has the same name.

Start simple: devmification

```
@prb@
identifier platform.probefn, pdev;
expression e, e1, e2;
@@
probefn(struct platform_device *pdev, ...) {
  <+...
- e = kzalloc(e1, e2)
+ e = devm_kzalloc(&pdev->dev, e1, e2)
  ...
?-kfree(e);
  ...+>
}

@rem depends on prb@
identifier platform.removefn;
expression prb.e;
@@
removefn(...) {
  <...
- kfree(e);
  ...>
}
```

Start simple: devmification

```
@prb@
identifier platform.probefn, pdev;
expression e, e1, e2;
@@
probefn(struct platform_device *pdev, ...) {
  <+...
- e = kzalloc(e1, e2)
+ e = devm_kzalloc(&pdev->dev, e1, e2)
  ...
?-kfree(e);
  ...+>
}

@rem depends on prb@
identifier platform.removefn;
expression e;
@@
removefn(...) {
  <...
- kfree(e);
  ...>
}
```

Same number of affected files. $\Longrightarrow$ Simple was good enough.

Incremental development

Restrict a semantic patch to reduce results.

- Devmification semantic patch returns results in 171 files.
  - Too many to study at once.
- Devmification may undesirably affect resource liberation order.
  - Implements LIFO order.
- $\Longrightarrow$ Rewrite the semantic patch to avoid this problem.

Adding restrictions on probe functions

- No value returning call prior to `kzalloc`.
- No call after `kfree`.

```
@prb@
identifier platform.probefn,pdev,f,g; expression e,e1,e2,x;
@@
probefn(struct platform_device *pdev, ...) {
  ... when != x = f(...);
- e = kzalloc(e1, e2)
+ e = devm_kzalloc(&pdev->dev, e1, e2)
  if(...) S
  ... when strict
(
 return 0;
|
- kfree(e);
)
  ... when != g(...);
}
```

Adding restrictions on probe functions

- No value returning call prior to `kzalloc`.
- No call after `kfree`.

```
@prb@
identifier platform.probefn,pdev,f,g; expression e,e1,e2,x;
@@
probefn(struct platform_device *pdev, ...) {
  ... when != x = f(...);
- e = kzalloc(e1, e2)
+ e = devm_kzalloc(&pdev->dev, e1, e2)
  if(...) S
  ... when strict
(
 return 0;
|
- kfree(e);
)
  ... when != g(...);
}
```

Results in 51 files.

Incremental development

Restrict a semantic patch to eliminate false positives.

- Locks on blocking function case.
- Lock status may be inaccurate when locking is conditional.
- Causes false positives.

Example: False positive

```
do {
  if (pHba->host) {
    ...
    spin_lock_irqsave(pHba->host->host_lock, flags);
  }
  ...
  if (pHba->host) {
    spin_unlock_irqrestore(pHba->host->host_lock, flags);
    ...
  }
} while (rcode == -ETIMEDOUT);
...
if (copy_from_user (msg, user_msg, size)) {
  ...
}
```

drivers/scsi/dpt_i2o.c

Adding restrictions on locks and unlocks

```
@badr@
position p;
@@
spin_unlock_irqrestore(...)
... when != spin_lock_irqsave(...)
copy_from_user@p(...)

@@
position p != badr.p;
@@
* spin_lock_irqsave(...)
  ... when != spin_unlock_irqrestore(...)
* copy_from_user@p(...)
```

Exploiting information from other tools

Checking constification is tedious and time consuming.

- Must compile all affected files.
- Ideally, would warn the user about potentially difficult cases.

Exploiting information from other tools

Checking constification is tedious and time consuming.

- Must compile all affected files.
- Ideally, would warn the user about potentially difficult cases.

Information to provide affected structure initializations:

- Are all fields function pointers?
- Are some structures of the same type already constified?
- Are all affected structures in compiled files?
  - Compare current file to set of .o files in a reference kernel.

Exploiting information from other tools

Checking constification is tedious and time consuming.

- Must compile all affected files.
- Ideally, would warn the user about potentially difficult cases.

Information to provide affected structure initializations:

- Are all fields function pointers?
- Are some structures of the same type already constified?
- Are all affected structures in compiled files?
  - Compare current file to set of .o files in a reference kernel.

Collect information using ocaml or python code.

Sample output

```
drivers/misc/sgi-xp/xp_main.c
   xpc_interface: good: 0, bad: 1, (all compiled)

drivers/misc/vexpress-syscfg.c
   vexpress_config_bridge_ops: good: 0, bad: 1

drivers/net/ethernet/hisilicon/hns/hnae.c
   hnae_buf_ops: good: 0, bad: 1, (done), (all compiled)
```

Conclusion

Coccinelle

- Patch-like language for searching and transforming an entire code base.
- Eases detection of project-specific issues.

Usage strategies

- Start simple.
- Refine as needed
  - To reduce number and diversity of results.
  - To eliminate false positives.
- Exploit external information to facilitate validation.

Compromises on both soundness and completeness,
but still useful in practice.

`http://coccinelle.lip6.fr/`