Prequel: A Patch-Like Query Language for Commit History Search

Julia Lawall, Derek Palinski, Gilles Muller
(Inria/LIP6)

August, 2016

Our focus: the Linux kernel

Linux is critical software.

- Used in embedded systems, desktops, servers, etc.

Linux is very large.

- Over 22 000 .c files
- Over 13.6 million lines of C code in Linux 4.4.
- Increase of 44% since July 2011 (Linux 3.0).

Linux has both more and less experienced developers.

– Maintainers, contributors, developers of proprietary drivers

Our focus: the Linux kernel

Linux is critical software.

- Used in embedded systems, desktops, servers, etc.

Linux is very large.

- Over 22 000 .c files
- Over 13.6 million lines of C code in Linux 4.4.
- Increase of 44% since July 2011 (Linux 3.0).

Linux has both more and less experienced developers.

- Maintainers, contributors, developers of proprietary drivers

Developers need reliable and precise information...

Our first effort: Coccinelle

Goal: Automating evolutions in C code

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: Automating evolutions in C code

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.

Example

Evolution: A new function: kzalloc (Linux 2.6.14)

⟹ Collateral evolution: Merge kmalloc and memset into kzalloc

```
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));
```

Example

Evolution: A new function: kzalloc (Linux 2.6.14)

⟹ Collateral evolution: Merge kmalloc and memset into kzalloc

```
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;
}
```

A kmalloc → kzalloc semantic patch

```
@@
expression x, E;
identifier f;
@@

x =
-   kmalloc
+   kzalloc
        (...)
  ...

- memset(x, 0, ...);
```

A kmalloc → kzalloc semantic patch

```
@@
expression x, E;
identifier f;
@@

x =
-    kmalloc
+    kzalloc
         (...)
  ...  when != (<+...x...+>) = E
       when != f(...,x,...)
- memset(x, 0, ...);
```

A kmalloc → kzalloc semantic patch

```
@@
expression x, E;
identifier f;
@@

x =
-   kmalloc
+   kzalloc
        (...)
  ...  when != (<+...x...+>) = E
       when != f(...,x,...)
- memset(x, 0, ...);
```

Updates 45 occurrences
(for 20, the allocated and zeroed sizes may differ)

The next step: Prequel

The next step: Prequel

We can see how the code is today...

The next step: Prequel

We can see how the code is today...
... but how did it get that way?

The next step: Prequel

We can see how the code is today...

... but how did it get that way?

... and what were the alternatives?

Example code development question

We know that kmalloc + memset can be converted to kzalloc.

$\Longrightarrow$ Can it be converted into anything else?

Example code development question

We know that kmalloc + memset can be converted to kzalloc.
⟹ Can it be converted into anything else?

- `git log -G kmalloc v3.0..v4.4`: 4076 results in 305 sec
- `git log -G memset v3.0..v4.4`: 6913 results in 296 sec
- `git log -S kmalloc v3.0..v4.4`: 3100 results in 212 sec
- `git log -S memset v3.0..v4.4`: 5662 results in 231 sec
- Don't know replacement function, so can't specify a line range for `git log -L` or `git blame`
- Don't know what commit to show.

Prequel idea

Find patches that closely resemble a patch query:

```
@@
expression x;

@@

  x =
-   kmalloc
+   kzalloc
       (...)
  ...
- memset(x, 0, ...);
```

Prequel idea

Or rather...

```
@@
expression x;
identifier f != kzalloc;
@@

  x =
-   kmalloc
+   f
        (...)
  ...
- memset(x, 0, ...);
```

Our proposal

Prequel: Patch Query Language

- Code-like pattern-based query language for describing patched code.
  - Builds on the Coccinelle pattern language (SmPL).

Our proposal

Prequel: Patch Query Language

- Code-like pattern-based query language for describing patched code.
  - Builds on the Coccinelle pattern language (SmPL).
- Approximate querying:
  - − and + lines must match changes.
  - Changes also allowed in matches of context lines.
  - Configurable with constraints on the amount of other changes in the commit.

Implementation strategy

How to match:

```
@@ expression x; identifier f != kzalloc; @@
  x =
-   kmalloc
+   f
      (...)
  ...
- memset(x, 0, ...);
```

against:

```
@@ -418 +418 @@ static void fnic_fcoe_process_vlan_resp(
- vlan = kmalloc(sizeof(*vlan),
+ vlan = kzalloc(sizeof(*vlan),
@@ -426 +426,0 @@ static void fnic_fcoe_process_vlan_resp(
- memset(vlan, 0, sizeof(struct fcoe_vlan));
```

Our solution: Reuse matching features of Ccocinelle

Split the patch query into:

- minus slice: – code and unannotated code.
- plus slice: + code and unannotated code.

Our solution: Reuse matching features of Ccocinelle

Split the patch query into:

- minus slice: - code and unannotated code.
- plus slice: + code and unannotated code.

Example:

```
@@
expression x;
identifier f != kzalloc;
@@
  x =
-   kmalloc
+   f
        (...)
  ...
- memset(x, 0, ...);
```

Our solution: Reuse matching features of Ccocinelle

Split the patch query into:

- minus slice: - code and unannotated code.
- plus slice: + code and unannotated code.

Minus slice:

```
@@
expression x;

@@
  x =
-   kmalloc

      (...)
  ...
- memset(x, 0, ...);
```

Plus slice:

```
@@
expression x;
identifier f != kzalloc;
@@
  x =

+   f
      (...)
```

Our solution

- Match minus slice against complete before files
- Match plus slice against complete after files

Our solution

- Match minus slice against complete before files
- Match plus slice against complete after files

Need to synchronize

- `x` should match the same term in before and after code.
- `kmalloc` and `kzalloc` should match tokens in the same hunk.
- Entire `memset` call should be in a hunk.

Generated code (simplified)

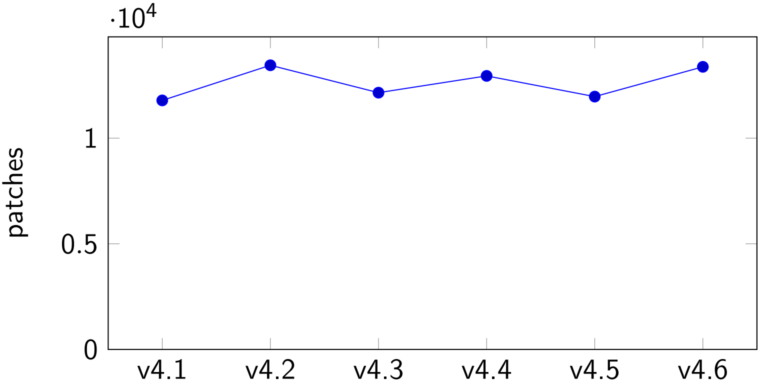
```
@r depends on before@
expression x;
position m1,m2,m3;
@@
x = kmalloc@m1(...)
...
memset@m2(x, 0, ...);@m3

@s depends on after@
expression r.x;
identifier f != kzalloc;
position p1;
@@
x = f@p1(...)

@script:ocaml@
m1 << r.m1; m2 << r.m2; m3 << r.m3; p1 << r.p1;
@@
match in_same_hunk_pair m1 m1 p1 p1 with
  Some hunkinfo1 ->
    (match in_same_hunk m2 m3 with
      Some hunkinfo2 -> output [hunkinfo1;hunkinfo2]
    | _ -> ())
| _ -> ()
```

Results: Linux v3.0..v4.4

Filtering options: –pct 0 –top 20 –all-lines

```
3d04fea: 100%
kcalloc
6d4ef68: 6%
kmalloc
1392402: 4%
rpcrdma_create_req
rpcrdma_create_rep
fe2fc9c: 4%
raw3215_alloc_info
180b138: 3%
kmalloc
10b1e0c: 0%
vmw_fifo_reserve
```

Scalability



Not practical to process all of them...

Filtering

Most patch queries don't need to be applied to all commits.

Filtering

Most patch queries don't need to be applied to all commits.

```
@@
expression x;
identifier f != kzalloc;
@@
  x =
-   kmalloc
+   f
       (...)
  ...
- memset(x, 0, ...);
```

Requires:

- Removed `kmalloc`
- Removed `memset`

Filtering

Idea:

- Search for `kmalloc` and `memset` in patches.
  - Much smaller than source files.
- Even better: make an index.

Filtering

Idea:

- Search for `kmalloc` and `memset` in patches.
  - Much smaller than source files.
- Even better: make an index.

Results:

- 270 selected commits
- 7 results
- $\sim$ 5 minutes

Another example

- `pn544_hci_i2c_probe` acquired an extra argument.
- What should the new value be?

Another example

- pn544_hci_i2c_probe acquired an extra argument.
- What should the new value be?

Using Prequel:

```
@@
expression e;
@@

pn544_hci_probe(...,
+ e,
  ...)
```

What filtering is possible?

Filtering alterative

```
@@
expression e;
@@

pn544_hci_probe(...,
+ e,
  ...)
```

– Changed tokens give no information.

+ `pn544_hci_probe` may be nearby.

+ Patches contain some context code.

Inferred criteria

- Patch must contain some added code.
- Patch must contain `pn544_hci_probe`, in context or changed lines.

Inferred criteria

- Patch must contain some added code.
- Patch must contain `pn544_hci_probe`, in context or changed lines.

Results:

- 7 commits analyzed.
- 1 result.

Inferred criteria

- Patch must contain some added code.
- Patch must contain pn544_hci_probe, in context or changed lines.

Results:

- 7 commits analyzed.
- 1 result.

Sample result:

```
  r = pn544_hci_probe(phy, &i2c_phy_ops, LLC_SHDLC_NAME,
                      PN544_I2C_FRAME_HEADROOM, PN544_I2C_FRAME_TAILROOM,
-                     PN544_HCI_I2C_LLC_MAX_PAYLOAD, &phy->hdev);
+                     PN544_HCI_I2C_LLC_MAX_PAYLOAD, NULL, &phy->hdev);
```

Inferred criteria

- Patch must contain some added code.
- Patch must contain pn544_hci_probe, in context or changed lines.

Results:

- 7 commits analyzed from Linux 3.0-4.6.
- 1 result.

Sample result:

```
  r = pn544_hci_probe(phy, &i2c_phy_ops, LLC_SHDLC_NAME,
                      PN544_I2C_FRAME_HEADROOM, PN544_I2C_FRAME_TAILROOM,
-                     PN544_HCI_I2C_LLC_MAX_PAYLOAD, &phy->hdev);
+                     PN544_HCI_I2C_LLC_MAX_PAYLOAD, NULL, &phy->hdev);
```

Risk of false negatives.

Another example

- `gpio_chip` structure has no `dev` field.
- How to access the needed information?

Another example

- gpio_chip structure has no dev field.
- How to access the needed information?

Using Prequel:

```
@@
struct gpio_chip *e;
@@
- e->dev
```

Another example

- gpio_chip structure has no dev field.
- How to access the needed information?

Using Prequel:

```
@@
struct gpio_chip *e;
@@
- e->dev
```

24,707 commits remove dev

Taking into account non-local information

Observations:

- Patch must contain `dev`
- Files may contain `gpio_chip`.

Taking into account non-local information

Observations:

- Patch must contain `dev`
- Files may contain `gpio_chip`.

Approach:

- Checking files of all patches would be costly.
- Grep for `gpio_chip` in a reference version.
- Check patches that affect those files.

Taking into account non-local information

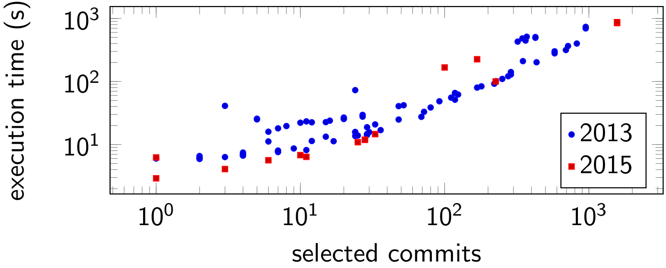
- 428 commits analyzed from Linux 3.0-4.6.
- 17 results.
- $\sim$15 minutes.
- Susceptible to false negatives

Evaluation

Driver porting:

- Dataset: 56 Linux drivers introduced in 2013 or 2015
- Compiled original versions in Linux v4.6 (May 2016)
- 103 patch queries to find commits illustrating fixes for the compiler errors.

Performance:



Comparison with git

	prior commits						prior lines					
	mn		avg		mx		mn		avg		max	
2013	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel
Unknown function	0	0	1.3	1.3	7	8	0	0	163.2	38.1	1421	239
Unknown type	0	0	0.0	0.0	0	0	0	0	0.0	0.0	0	0
Unknown field	3	0	852.7	0.7	2195	10	184	0	146K.1	63.1	411K	756
Unknown variable	0	0	2.6	0.0	6	0	0	0	202.6	0.0	452	0
Arg error	0	0	8.8	1.2	∞	10	0	0	1189.2	117.9	∞	943
Value type change	2	0	46.6	0.0	73	0	178	0	5691.9	0.0	8582	0
Context type change	∞	0	∞	0.1	∞	1	∞	0	∞	3.6	∞	29
2015	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel
Unknown function	0	0	0.5	0.0	2	0	0	0	87.8	0.0	351	0
Unknown field	217	0	996.0	0.6	2243	1	31K	0	143K	15.6	318K	29
Arg error	1	0	1.0	0.0	1	0	30	0	30.0	0.0	30	0

Comparison with git

	prior commits						prior lines					
	mn		avg		mx		mn		avg		max	
2013	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel
Unknown function	0	0	1.3	1.3	7	8	0	0	163.2	38.1	1421	239
Unknown type	0	0	0.0	0.0	0	0	0	0	0.0	0.0	0	0
Unknown field	3	0	852.7	0.7	2195	10	184	0	146K.1	63.1	411K	756
Unknown variable	0	0	2.6	0.0	6	0	0	0	202.6	0.0	452	0
Arg error	0	0	8.8	1.2	∞	10	0	0	1189.2	117.9	∞	943
Value type change	2	0	46.6	0.0	73	0	178	0	5691.9	0.0	8582	0
Context type change	∞	0	∞	0.1	∞	1	∞	0	∞	3.6	∞	29
2015	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel
Unknown function	0	0	0.5	0.0	2	0	0	0	87.8	0.0	351	0
Unknown field	217	0	996.0	0.6	2243	1	31K	0	143K	15.6	318K	29
Arg error	1	0	1.0	0.0	1	0	30	0	30.0	0.0	30	0

Comparison with git

	prior commits						prior lines					
	mn		avg		mx		mn		avg		max	
2013	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel
Unknown function	0	0	1.3	1.3	7	8	0	0	163.2	38.1	1421	239
Unknown type	0	0	0.0	0.0	0	0	0	0	0.0	0.0	0	0
Unknown field	3	0	852.7	0.7	2195	10	184	0	146K.1	63.1	411K	756
Unknown variable	0	0	2.6	0.0	6	0	0	0	202.6	0.0	452	0
Arg error	0	0	8.8	1.2	∞	10	0	0	1189.2	117.9	∞	943
Value type change	2	0	46.6	0.0	73	0	178	0	5691.9	0.0	8582	0
Context type change	∞	0	∞	0.1	∞	1	∞	0	∞	3.6	∞	29
2015	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel
Unknown function	0	0	0.5	0.0	2	0	0	0	87.8	0.0	351	0
Unknown field	217	0	996.0	0.6	2243	1	31K	0	143K	15.6	318K	29
Arg error	1	0	1.0	0.0	1	0	30	0	30.0	0.0	30	0

Comparison with git

	prior commits						prior lines					
	mn		avg		mx		mn		avg		max	
2013	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel
Unknown function	0	0	1.3	1.3	7	8	0	0	163.2	38.1	1421	239
Unknown type	0	0	0.0	0.0	0	0	0	0	0.0	0.0	0	0
Unknown field	3	0	852.7	0.7	2195	10	184	0	146K.1	63.1	411K	756
Unknown variable	0	0	2.6	0.0	6	0	0	0	202.6	0.0	452	0
Arg error	0	0	8.8	1.2	∞	10	0	0	1189.2	117.9	∞	943
Value type change	2	0	46.6	0.0	73	0	178	0	5691.9	0.0	8582	0
Context type change	∞	0	∞	0.1	∞	1	∞	0	∞	3.6	∞	29
2015	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel	git	prequel
Unknown function	0	0	0.5	0.0	2	0	0	0	87.8	0.0	351	0
Unknown field	217	0	996.0	0.6	2243	1	31K	0	143K	15.6	318K	29
Arg error	1	0	1.0	0.0	1	0	30	0	30.0	0.0	30	0

Conclusion

- Pattern language for searching commit histories
  - Patterns include context information.
  - User controls rate of unmatched changes.
- Potential applications:
  - How to modernize old code
  - Finding potentially risky constructions
  - Automated bug fixing
  - Metrics
- Moderately efficient
  - Under 30 seconds in 60% of driver examples.
- Available soon...

Prequel performance by error type.

	Total time (sec.)			Commit selection time			# commits selected			Cocci time per commit		
2013	mn	avg	mx	mn	avg	mx	mn	avg	mx	mn	avg	mx
Unknown function	6	21	66	4	5	8	2	22	118	0	1	11
Unknown type	6	6	6	5	5	5	1	1	1	0	0	0
Unknown field	11	299	736	4	86	292	6	373	951	0	1	2
Unknown variable	7	15	28	4	5	5	4	24	69	0	1	2
Arg error	6	35	316	4	6	24	2	66	693	0	0	1
Value type change	6	30	92	4	6	10	2	43	221	0	1	1
Context type change	49	93	141	7	10	12	92	201	288	0	0	0
2015	mn	avg	mx	mn	avg	mx	mn	avg	mx	mn	avg	mx
Unknown function	4	6	7	2	2	3	1	6	11	0	1	3
Unknown field	100	443	875	4	59	147	100	726	1569	0	0	0
Arg error	3	10	15	2	3	3	1	23	33	0	0	0

mn = min, avg = average, mx = max. Times rounded to the nearest second.