Getting Started with Coccinelle
KVM edition
part 1

Julia Lawall (Inria/LIP6/Irill/UPMC)

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

August 20, 2015

This tutorial is designed to work with Qemu version 2.4.0 (http://wiki.qemu.org/Download) and with Coccinelle version 1.0.0.

Other versions of Coccinelle should be acceptable.

Common programming problems

- The software evolution dilemma.
  - Modernizing APIs can make code more effective.
  - Updating all parts of the code is error prone and boring.
  - Mixing different functions for the same purpose is confusing.
- The software robustness problem.
  - Error conditions are not always checked for.
  - Error conditions may be checked for when no errors can occur.
- Programmers are sloppy.

Need for pervasive code changes.

Example: Overlapping APIs

```
void *cpu_physical_memory_map(hwaddr addr, hwaddr *plen,
                              int is_write)
{
  return address_space_map(&address_space_memory, addr, plen,
                           is_write);
}

void cpu_physical_memory_unmap(void *buffer, hwaddr len,
                               int is_write, hwaddr access_len)
{
  return address_space_unmap(&address_space_memory, buffer,
                             len, is_write, access_len);
}
```

Example: Overlapping APIs

```
void *cpu_physical_memory_map(hwaddr addr, hwaddr *plen,
                              int is_write)
{
  return address_space_map(&address_space_memory, addr, plen,
                           is_write);
}

void cpu_physical_memory_unmap(void *buffer, hwaddr len,
                               int is_write, hwaddr access_len)
{
  return address_space_unmap(&address_space_memory, buffer,
                             len, is_write, access_len);
}
```

Qemu BiteSizedTasks suggests using `address_space_*`:

- 28 occurrences of `cpu_physical_memory_map`, in 15 files.
- 38 occurrences of `cpu_physical_memory_unmap`, in 15 files.

Example: Incorrect error checking

```
ifd = open(argv[1], O_RDONLY);
if (ifd < 0) {
  fprintf(stderr, "%s: Can’t open %s: %s\n",
    argv[0], argv[1], strerror(errno));
  exit(EXIT_FAILURE);
}

ofd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, FILE_PERM);
if (ifd < 0) {
  fprintf(stderr, "%s: Can’t open %s: %s\n",
    argv[0], argv[2], strerror(errno));
  if (ifd)
    close(ifd);
  exit(EXIT_FAILURE);
}
```

roms/u-boot/board/samsung/origen/tools/mkorigenspl.c

Example: Incorrect error checking

```
ifd = open(argv[1], O_RDONLY);
if (ifd < 0) {
  fprintf(stderr, "%s: Can't open %s: %s\n",
    argv[0], argv[1], strerror(errno));
  exit(EXIT_FAILURE);
}

ofd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, FILE_PERM);
if (ifd < 0) {
  fprintf(stderr, "%s: Can't open %s: %s\n",
    argv[0], argv[2], strerror(errno));
  if (ifd)
    close(ifd);
  exit(EXIT_FAILURE);
}
```

roms/u-boot/board/samsung/origen/tools/mkorigenspl.c

Example: Complicated code that does very little

```
int power_init_board(void)
{
  int ret;

  /*
   * For PMIC the I2C bus is named as I2C5, but it
   * is connected to logical I2C adapter 0
   */
  ret = pmic_init(I2C_0);
  if (ret)
    return ret;

  return 0;
}
```

roms/u-boot/board/samsung/goni/goni.c

Our goals

- Automatically find code containing bugs or defects, or requiring evolutions.
- Automatically fix bugs or defects, and perform evolutions.
- Provide a system that is accessible to software developers.

Requirements for automation

The ability to abstract over irrelevant information:

- `cpu_physical_memory_unmap(data, size, 0, 0);`
- Argument values don't matter.

The ability to transform code fragments:

- `ofd = ...; if (ifd < 0) ...`
- Replace `ifd < 0` by `ofd < 0`.

The ability to match scattered code fragments:

- A newly unnecessary declaration may be far from its former use.

Coccinelle

Program matching and transformation for unpreprocessed C code.

Fits with the existing habits of C programmers.

- C-like, patch-like notation

Semantic patch language (SmPL):

- Metavariables for abstracting over subterms.
- "..." for abstracting over code sequences.
- Patch-like notation ($-/+$) for expressing transformations.
- $*$ for searching.

Example: cpu_physical_memory_* functions

Our goal:

Replace the cpu functions by the address space functions.

```
void *cpu_physical_memory_map(hwaddr addr, hwaddr *plen,
                              int is_write)
{
  return address_space_map(&address_space_memory, addr, plen,
                           is_write);
}

void cpu_physical_memory_unmap(void *buffer, hwaddr len,
                               int is_write, hwaddr access_len)
{
  return address_space_unmap(&address_space_memory, buffer,
                             len, is_write, access_len);
}
```

Building a semantic patch

Select some model code fragments:

```
cpu_physical_memory_map(addr, &size, 0)
```

```
cpu_physical_memory_unmap(data, size, 0, 0)
```

Building a semantic patch

Generalize and write the corresponding transformation:

```
cpu_physical_memory_map(addr, plen, is_write)
```

```
cpu_physical_memory_unmap(buffer, len, is_write, access_len)
```

Building a semantic patch

Generalize and write the corresponding transformation:

```
- cpu_physical_memory_map(addr, plen, is_write)
+ address_space_map(&address_space_memory, addr, plen, is_write)
```

```
-  cpu_physical_memory_unmap(buffer, len, is_write, access_len)
+ address_space_unmap(&address_space_memory, buffer, len,
                      is_write, access_len)
```

Building a semantic patch

Add metavariable declarations:

```
@@
expression addr, plen, is_write;
@@
- cpu_physical_memory_map(addr, plen, is_write)
+ address_space_map(&address_space_memory, addr, plen, is_write)

@@
expression buffer, len, is_write, access_len;
@@
- cpu_physical_memory_unmap(buffer, len, is_write, access_len)
+ address_space_unmap(&address_space_memory, buffer, len,
+                     is_write, access_len)
```

Building a semantic patch

Add metavariable declarations:

```
@@
expression addr, plen, is_write;
@@
- cpu_physical_memory_map(addr, plen, is_write)
+ address_space_map(&address_space_memory, addr, plen, is_write)

@@
expression buffer, len, is_write, access_len;
@@
- cpu_physical_memory_unmap(buffer, len, is_write, access_len)
+ address_space_unmap(&address_space_memory, buffer, len,
+                      is_write, access_len)
```

Modifies 62 calls

Semantic patch structure

Two parts per rule:

- Metavariable declaration
- Transformation specification

A semantic patch can contain multiple rules.

Metavariable types

- expression, statement, identifier, type, constant, local idexpression
- A type from the source program
- iterator, declarer, iterator name, declarer name, typedef

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 irrelevant.

Exercise 1

1. Create a file cpu.cocci containing the following:

```
@@
expression addr, plen, is_write;
@@
- cpu_physical_memory_map(addr, plen, is_write)
+ address_space_map(&address_space_memory, addr, plen, is_write)

@@
expression buffer, len, is_write, access_len;
@@
- cpu_physical_memory_unmap(buffer, len, is_write, access_len)
+ address_space_unmap(&address_space_memory, buffer, len,
+                     is_write, access_len)
```

2. Run spatch: `spatch --sp-file cpu.cocci --dir qemu/hw/virtio`
3. Did your semantic patch do everything it should have?
4. Did it do something it should not have?

Exercise 2

1. When using Coccinelle, - and + can be placed on any token, not necessarily complete terms. Compare your previous result with the result for the following rule (unmap case omitted for conciseness):

   ```
   @@
   expression addr, plen, is_write;
   @@
   - cpu_physical_memory_map(
   + address_space_map(&address_space_memory,
                               addr, plen, is_write)
   ```

2. Likewise, try the following transformation rule:

   ```
   @@
   @@
   - cpu_physical_memory_map(
   + address_space_map(&address_space_memory,
   ```

Exercise 3

The following code is unnecessarily complex, in that the result of calling `ubi_io_write` could just be returned directly:

```
err = ubi_io_write(ubi, p, pnum, ubi->vid_hdr_aloffset,
                   ubi->vid_hdr_alsize);
return err;
```

1. Write a semantic patch to simplify such code and test it on `roms/u-boot/drivers/usb`.
2. Are all of the transformations correct?
3. In writing this semantic patch, `err` can be represented as a metavariable that is declared to be an `expression`, an `identifier`, or an `idexpression`. Try these three options and see which gives the best results. (Hint: look at roms/u-boot/drivers/usb/gadget/composite.c and roms/u-boot/drivers/dfu/dfu.c)

Practical issues

To check that a semantic patch is valid:

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

To run a semantic patch:

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

Put the interesting output in a file:

```
spatch ... > output.patch
```

Omit the uninteresting output:

```
spatch --very-quiet ...
```

More practical issues

If you don't need to include header files:

```
spatch --sp-file mysp.cocci --dir directory
                 --no-includes --include-headers
```

To understand why your semantic patch didn't work:

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

The source code:

`qemu-2.4.0`, from `http://wiki.qemu.org/Download`

These slides:

http://events.linuxfoundation.org/events/kvm-forum/program/slides

Example: Incorrect error checking

```
ofd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, FILE_PERM);
if (ifd < 0) {
  fprintf(stderr, "%s: Can't open %s: %s\n",
    argv[0], argv[2], strerror(errno));
  if (ifd)
    close(ifd);
  exit(EXIT_FAILURE);
}
```

Example: Incorrect error checking

```
ofd = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, FILE_PERM);
if (ifd < 0) {
  fprintf(stderr, "%s: Can't open %s: %s\n",
    argv[0], argv[2], strerror(errno));
  if (ifd)
    close(ifd);
  exit(EXIT_FAILURE);
}
```

Problem: The tested variable is different than the assigned one.

Semantic patch attempt

```
@@
expression x, y;
identifier f;
statement S;
@@

x = f(...);
if (y < 0) S
```

Semantic patch attempt

```
@@
expression x, y;
identifier f;
statement S;
@@

x = f(...);
if (
- y
+ x
  < 0) S
```

Semantic patch attempt

```
@@
expression x, y;
identifier f;
statement S;
@@

x = f(...);
if (
- y
+ x
  < 0) S
```

Does it work? Does it seem like the best way to do it?

Semantic patch attempt

Problem:

- `expression x` and `expression y` match all expressions.
- Both same expressions and different expressions.

Semantic patch attempt

Problem:

- `expression x` and `expression y` match all expressions.
- Both same expressions and different expressions.

Two cases:

```
x = f(...);
if (x < 0) S
```

and:

```
x = f(...);
if (y < 0) S
```

where `y` is different from `x`.

Semantic patch attempt

Problem:

- `expression x` and `expression y` match all expressions.
- Both same expressions and different expressions.

Two cases:

```
x = f(...);
if (x < 0) S
```

and:

```
x = f(...);
if (y < 0) S
```

where `y` is different from `x`.

Want to transform only the second one.

Disjunction

- A sequence of patterns between `( ...| ... )`.
- Patterns checked in order and the first that matches is chosen.
- Must be escaped (\) if not in column 0.

```
@@
identifier x,y;
identifier f;
statement S;
@@
  x = f(...);
(
  if (x < 0) S
|
  if (
-     y
+     x
      < 0) S
)
```

Exercise 4

Qemu coding style encourages the use of braces around all if branches. One way to achieve this would be (single-branch case only):

```
@@
expression e; statement S;
@@
if (e)
+ {
S
+ }
```

But this rule adds braces around if branches that already have braces, because an open brace followed by code followed by a close branch represents a single statement.

Use a disjunction to write a single rule that puts braces around some common if branch cases, such as an assignment, break, return, etc. Test your rule on dtc/fdtdump.c.

[Hint on next page]

Exercise 4, contd.

`Hint:` + code has to be attached to an actual existing or removed token; it cannot be attached to the outside of a disjunction.

Compressing return sequences

```
int power_init_board(void)
{
  int ret;

  /*
   * For PMIC the I2C bus is named as I2C5, but it
   * is connected to logical I2C adapter 0
   */
  ret = pmic_init(I2C_0);
  if (ret)
    return ret;

  return 0;
}
```

Compressing return sequences

```
int power_init_board(void)
{
  int ret;

  /*
   * For PMIC the I2C bus is named as I2C5, but it
   * is connected to logical I2C adapter 0
   */
  ret = pmic_init(I2C_0);
  if (ret)
    return ret;

  return 0;
}
```

Compressing return sequences

```
int power_init_board(void)
{
  int ret;

  /*
   * For PMIC the I2C bus is named as I2C5, but it
   * is connected to logical I2C adapter 0
   */
  ret = pmic_init(I2C_0);
  return ret;



}
```

Compressing return sequences

```
int power_init_board(void)
{
  int ret;

  /*
   * For PMIC the I2C bus is named as I2C5, but it
   * is connected to logical I2C adapter 0
   */
  ret = pmic_init(I2C_0);
  return ret;



}
```

Compressing return sequences

```
int power_init_board(void)
{
  int ret;

  /*
   * For PMIC the I2C bus is named as I2C5, but it
   * is connected to logical I2C adapter 0
   */
  return pmic_init(I2C_0);




}
```

Compressing return sequences

```
int power_init_board(void)
{
  int ret;

  /*
   * For PMIC the I2C bus is named as I2C5, but it
   * is connected to logical I2C adapter 0
   */
  return pmic_init(I2C_0);





}
```

Compressing return sequences

```
int power_init_board(void)
{


  /*
   * For PMIC the I2C bus is named as I2C5, but it
   * is connected to logical I2C adapter 0
   */
  return pmic_init(I2C_0);




}
```

Some semantic patch rules

```
@@

@@
- if (ret) return ret;
- return 0;
+ return ret;

@@

@@
- ret = e;
- return ret;
+ return e;
```

Some semantic patch rules

```
@@
expression ret;
@@
- if (ret) return ret;
- return 0;
+ return ret;

@@
expression ret, e;
@@
- ret = e;
- return ret;
+ return e;
```

False positive

```
- c->rate = ccu_clk->freq_tbl[ccu_clk->freq_id];
- return c->rate;
+ return ccu_clk->freq_tbl[ccu_clk->freq_id];
```

False positive

```
- c->rate = ccu_clk->freq_tbl[ccu_clk->freq_id];
- return c->rate;
+ return ccu_clk->freq_tbl[ccu_clk->freq_id];
```

For this rule:

```
@@
expression ret, e;
@@
- ret = e;
- return ret;
+ return e;
```

`ret` cannot be an arbitrary expression

Revised semantic patch rules

```
@@
expression ret;
@@
- if (ret) return ret;
- return 0;
+ return ret;

@@
local idexpression ret;
expression e;
@@
- ret = e;
- return ret;
+ return e;
```

Removing unused identifiers

The following rule may remove the only use of `ret`:

```
@@
local idexpression ret;
expression e;
@@
- ret = e;
- return ret;
+ return e;
```

Problem: The declaration of `ret` can be far from the use.

Dots

```
@@
type T;
identifier i;
@@
- T i;
  ... when != i
```

Dots

```
@@
type T;
identifier i;
@@
- T i;
  ... when != i
```

"..." matches all possible execution paths from the pattern before to the pattern after

- No pattern before means the beginning of the function.
- No pattern after means the end of the function.
- The patterns before and after cannot appear in the region matched by "..." (shortest path principle).

The complete semantic patch

```
@@
expression ret;
@@
- if (ret) return ret;
- return 0;
+ return ret;

@@
local idexpression ret;
expression e;
@@
- ret = e;
- return ret;
+ return e;

@@
type T; identifier i;
@@
- T i;
  ... when != i
```

Example

```
int power_init_board(void)
{
-  int ret;
-
  /*
   * For PMIC the I2C bus is named as I2C5, but it
   * is connected to logical I2C adapter 0
   */
-  ret = pmic_init(I2C_0);
-  if (ret)
-    return ret;
-
-  return 0;
+  return pmic_init(I2C_0);
}
```

Exercise 5

The semantic patch for removing unused variables only matches a variable declaration when the declaration does not initialize the variable.

- Extend the complete semantic patch (slide 52), so that it also removes unused variables that are initialized to a constant.
- Test your semantic patch on hw/display and hw/xen.
- Another issue is that a declaration may declare multiple variables. What does Coccinelle do in this case?
- **Hint:** A metavariable declared as `constant` will only match a constant.

Exercise 6

We have seen the use of "..." to match a sequence of statements. "..." can be used to match other kinds of terms whose identity doesn't matter, such as a sequence of arguments. Use "..." and a disjunction to write a single semantic patch rule that addresses (some part of) the following Qemu code transition:

> *The QError macros QERR_ are a hack to help with the transition to the current error.h API. Avoid them in new code, use simple message strings instead.*

Some possible transformations are:

```
- error_setg(errp, QERR_IO_ERROR);
+ error_setg(errp, "An IO error has occurred");

- error_setg(errp, QERR_DEVICE_HAS_NO_MEDIUM, device);
+ error_setg(errp, "Device '%s' has no medium", device);

- error_setg(errp, QERR_PROPERTY_VALUE_BAD,
+ error_setg(errp, "Property '%s.%s' doesn't take value '%s'",
             object_get_typename(OBJECT(dev)), prop->name, value);
```

Summary

SmPL features seen so far:

- Metavariables for abstracting over arbitrary terms.
- Metavariables restricted to particular types.
- Multiple rules.
- Disjunctions.
- Dots.

Getting Started with Coccinelle

KVM edition

part 2

Julia Lawall (Inria/LIP6/Irill/UPMC)

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

August 20, 2015

Malloc bite-sized task

- Convert uses of malloc to either g_malloc, g_new
  - More rarely g_try_malloc or g_try_new if a lot of memory is being allocated.
- Likewise, convert calloc to either g_new0 or g_try_new0.
- Drop return value checks unless using g_try_new/g_try_new0.

g_malloc man page

- These functions provide support for allocating and freeing memory.
  - If any call to allocate memory fails, the application is terminated. There is no need to check if the call succeeded.
  - It's important to match g_malloc() (and wrappers such as g_new()) with g_free()
- `g_malloc`: Allocates n_bytes bytes of memory. If n_bytes is 0 it returns NULL.
- `g_new`: Allocates n_structs elements of type struct_type.
  - The returned pointer is cast to a pointer to the given type: it is normally unnecessary to cast it explicitly.
  - If n_structs is 0 it returns NULL.

Issues we consider

- `g_malloc`/`g_new` should match with `g_free`
- `g_new` should be used for structures.
- Casts on the result of `g_new` are not needed (exercise).
- NULL tests are not needed.

Issues we ignore

- Large allocations should use `try` functions.
  - Coccinelle doesn't know much about values.
  - Could make a rule that highlights cases that need attention.
- For a calculated size, `g_malloc/g_new` may return 0.
  - Coccinelle doesn't know much about values.
  - Could make a rule that highlights cases that need attention.
- `g_malloc` vs. `g_malloc0`
  - Left to an exercise.

Basic malloc transformation

```
@@
expression e;
@@

- malloc(e)
+ g_malloc(e)
```

Example

block/iscsi.c:

```
- acb->task = malloc(sizeof(struct scsi_task));
+ acb->task = g_malloc(sizeof(struct scsi_task));
```

Changes 836 calls in qemu-2.4.0-rc4, in 428 files.

Malloc transformation assessment

+ There are many occurrences.
  - Lot of opportunity for automation.

– How do we find the corresponding frees?
  - Convert all mallocs and frees, and hope for the best?
  - Find mallocs that are always followed by frees?
  - Something else?

Malloc transformation assessment

+ There are many occurrences.
    - Lot of opportunity for automation.

– How do we find the corresponding frees?
    - Convert all mallocs and frees, and hope for the best?
    - Find mallocs that are always followed by frees?
    - Something else?

    We try the first option...

Malloc and free: first attempt

```
@@
expression x,e;
@@

- x = malloc(e)
+ x = g_malloc(e)
  ...
- free(x)
+ g_free(x)
```

Malloc and free: safer version

```
@@
expression x,e,e1;
@@

- x = malloc(e)
+ x = g_malloc(e)
  ... when != x = e1
- free(x)
+ g_free(x)
```

Malloc and free: Potential for false positives

"`...`" matches all paths except error paths.

Example:

```
@@
@@
a();
...
b();
...
c();
```

Malloc and free: Potential for false positives

"..." matches all paths except error paths.

Example:

```
@@
@@
a();
...
b();
...
c();
```

`c()` may be missing if `a()` or `b()` fails.

Malloc and free: False positive

roms/u-boot/fs/yaffs2/yaffs_nandif.c:

```
YCHAR *clonedName = malloc(...);
struct yaffs_dev *dev = malloc(...);
struct yaffs_param *param;

if (dev && clonedName) {
  memset(dev, 0, sizeof(struct yaffs_dev));
  strcpy(clonedName, name);
  param = &dev->param;
  param->name = clonedName;
  ...
  return dev;
}
free(dev);
free(clonedName);
```

Malloc and free: False positive

roms/u-boot/fs/yaffs2/yaffs_nandif.c:

```
YCHAR *clonedName = malloc(...);
struct yaffs_dev *dev = malloc(...);
struct yaffs_param *param;

if (dev && clonedName) {
  memset(dev, 0, sizeof(struct yaffs_dev));
  strcpy(clonedName, name);
  param = &dev->param;
  param->name = clonedName;
  ...
  return dev;
}
free(dev);
free(clonedName);
```

Success path in the if, failure path afterwards.

How to find cases where malloc'd data is always freed

Between malloc and free there can be many ifs.

- Some test for failure of malloc and abort
- Some test for failure of other things and abort
- Some do not abort

How to find cases where malloc'd data is always freed

Between malloc and free there can be many ifs.

- Some test for failure of malloc and abort

  No free expected.

- Some test for failure of other things and abort

- Some do not abort

How to find cases where malloc'd data is always freed

Between malloc and free there can be many ifs.

- Some test for failure of malloc and abort

  No free expected.

- Some test for failure of other things and abort

  Free required.

- Some do not abort

How to find cases where malloc'd data is always freed

Between malloc and free there can be many ifs.

- Some test for failure of malloc and abort

  No free expected.
- Some test for failure of other things and abort

  Free required.
- Some do not abort

  Any behavior is acceptable.

How to find cases where malloc'd data is always freed

Between malloc and free there can be many ifs.

- Some test for failure of malloc and abort

  No free expected.

- Some test for failure of other things and abort

  Free required.

- Some do not abort

  Any behavior is acceptable.

Issue: We don't know how many times these things will occur.

Improved malloc and free transformation

```
@@ expression x,e; @@

- x = malloc(e)
+ x = g_malloc(e)








- free(x)
+ g_free(x)
```

Improved malloc and free transformation

Adding a nest:

```
@@ expression x,e; @@

- x = malloc(e)
+ x = g_malloc(e)
  <...







  ...>
- free(x)
+ g_free(x)
```

Improved malloc and free transformation

Constraining a nest (forbidden code):

```
@@ expression x,e; @@

- x = malloc(e)
+ x = g_malloc(e)
  <... when != if (...) { ... return ...; }
       when != x = e1









  ...>
- free(x)
+ g_free(x)
```

Improved malloc and free transformation

Adding exceptions (allowed code):

```
@@ expression x,e; statement S; @@

- x = malloc(e)
+ x = g_malloc(e)
  <... when != if (...) { ... return ...; }
       when != x = e1
(
  if (x == NULL) S
|
  if (...) { ... free(x); ... return ...; }
)
  ...>
- free(x)
+ g_free(x)
```

Transforms 108 out of 836 occurrences.

Exercise 7

The basic malloc transformation semantic patch (slide 6) can also be written as:

```
@@
expression e;
@@

- malloc
+ g_malloc
       (e)
```

- Implement both strategies and try them on the pixmax/pixman directory.
- What is the difference between the results?

Exercise 8

Test the original malloc-free semantic patch:

```
@@ expression x,e; @@
- x = malloc(e)
+ x = g_malloc(e)
  ...
- free(x)
+ g_free(x)
```

on the following files:

- pixman/test/utils.c
- roms/u-boot/board/zeus/zeus.c
- roms/openbios/arch/sparc32/multiboot.c

Are all cases false positives (malloc of a non-local buffer)?
Or does Qemu contain some potential memory leaks?

Exercise 9

Often when allocating memory it is necessary to also zero it, as exemplified by the following code (roms/u-boot/fs/zfs/zfs.c):

```
data = malloc(sizeof(*data));
if (!data)
  return 0;
memset(data, 0, sizeof(*data));
```

`g_malloc` has a counterpart, `g_malloc0`, that does this directly.

Pretend that there exists a function `malloc0` that both mallocs and zeroes, and write a semantic patch that introduces uses of this function.

[Continued on the next page]

Exercise 9, contd

For example, your semantic patch should produce the following result on the above code:

```
data = malloc0(sizeof(*data));
if (!data)
  return 0;
```

Try your semantic patch on roms/u-boot/drivers/net.

Are there any false positives? Can you imagine how there could be any? If you don't find any, rewrite your semantic patch to convert `g_malloc` to `g_malloc0` and assess the results.

Using g_new

g_new: Allocates n_structs elements of type struct_type.

Easy case: Structure type explicit:

```
@@
identifier s;
@@

- g_malloc(sizeof(struct s))
+ g_new(s,1)
```

Using g_new

g_new: Allocates n_structs elements of type struct_type.

Easy case: Structure type explicit:

```
@@
identifier s;
@@

- g_malloc(sizeof(struct s))
+ g_new(s,1)
```

Six occurrences

Other possibilities for `g_new`

- Zeroed allocation (exercise):

  ```
  g_malloc0(sizeof(struct omap_mmc_s))
  ```

- Array allocation:

  ```
  g_malloc0(sizeof(struct iovec) * ab->nr_entries)
  ```

- Sizeof expression: `g_malloc(sizeof(*config))`
- Sizeof typedef: `g_malloc0(sizeof(QEMUTimer))`
- Combination: `g_malloc(niov * sizeof(*iov))`

Array allocation

Example:

```
- *iov = g_malloc0(sizeof(struct iovec) * ab->nr_entries);
+ *iov = g_new0(iovec, ab->nr_entries);
```

Array allocation

Example:

```
- *iov = g_malloc0(sizeof(struct iovec) * ab->nr_entries);
+ *iov = g_new0(iovec, ab->nr_entries);
```

Semantic patch:

```
@@
identifier i;
expression e;
@@

- g_malloc0(sizeof(struct i) * e)
+ g_new0(i, e)
```

Results

Five `g_malloc0` calls transformed

As expected:

```
- *iov = g_malloc0(sizeof(struct iovec) * ab->nr_entries);
+ *iov = g_new0(iovec, ab->nr_entries);
```

Results

Five `g_malloc0` calls transformed

As expected:

```
- *iov = g_malloc0(sizeof(struct iovec) * ab->nr_entries);
+ *iov = g_new0(iovec, ab->nr_entries);
```

Perhaps unexpected:

```
- s->modules = g_malloc0(s->modulecount *
-                         sizeof(struct omap2_gpio_s));
+ s->modules = g_new0(omap2_gpio_s, s->modulecount);
```

Isomorphisms

Issues:

- Coccinelle matches code exactly as it appears.
- `sizeof(struct i) * e` does not match `e * sizeof(struct i)`.
- Leads to rule duplication for trivial variants.

Isomorphisms

Issues:

- Coccinelle matches code exactly as it appears.
- `sizeof(struct i) * e` does not match `e * sizeof(struct i)`.
- Leads to rule duplication for trivial variants.

Isomorphisms:

- Transparently treat similar code patterns in a similar way.

Isomorphisms

Defined in standard.iso:

```
Expression
@ mult_comm @
expression X, Y;
@@
X * Y => Y * X
```

Effect visible using: spatch --parse-cocci sp.cocci:

```
(
-g_malloc0
  >>> g_new0(r:i, r:e)
-(-sizeof-(-struct -r:i -) -* -r:e-)
|
-g_malloc0
  >>> g_new0(r:i, r:e)
-(-r:e -* -sizeof-(-struct -r:i -)-)
)
```

Some other useful Isomorphisms

```
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
```

Exercise 10

g_malloc etc. abort the program if the allocation fails, and thus testing the result is not needed. Write a semantic patch that removes such NULL tests that immediately follow calls to g_malloc and g_malloc0. For example, your semantic patch should perform the following transformation on linux-user/elfload.c:

```
    info->notes = g_malloc0(NUMNOTES *
                            sizeof (struct memelfnote));
  - if (info->notes == NULL)
  -   return (-ENOMEM);
```

Test your semantic patch on

- linux-user/elfload.c
- hw/net

What isomorphisms are involved in doing these transformations?

Exercise 11

If we consider malloc, rather than g_malloc etc., there are cases such as the following, from roms/u-boot/common/env_attr.c, where the NULL testing if has both "then" and "else" branches.

```
entry_cpy = malloc(entry_len + 1);
if (entry_cpy)
  /* copy the rest of the list */
  strcpy(entry_cpy, entry);
else
  return -ENOMEM;
```

- Create a semantic patch that will remove the if test and the appropriate branch (for testing purposes, your semantic patch should apply to calls to malloc).

[Continued on the next page]

Exercise 11, contd.

- Test your semantic patch on roms/u-boot. Is the result satisfactory? If not, try to improve it.
- If the NULL test has both “then” and “else” branches, it may be that it is possible to recover from failure. In this case, `g_try_malloc`, etc. should be used instead of `g_malloc`, etc. How could you characterize such conditions.

Exercise 12

g_new and g_new0 are macros that cast their result to the type named in their first argument. Thus, no cast is needed on the result.

- Write a semantic patch that transforms calls to g_malloc0 that have a structure name in the first argument and that have a cast on the result to appropriate calls to g_new0. An example, from hw/misc/omap_sdrc.c is as follows:

```
- struct omap_sdrc_s *s = (struct omap_sdrc_s *)
-         g_malloc0(sizeof(struct omap_sdrc_s));
+ struct omap_sdrc_s *s = g_new0(omap_sdrc_s, 1);
```

- Test your semantic patch on hw/misc
- Does anything change that you did not expect? If so, adjust your semantic patch to only change what was intended.

Sizeof expression

Problem: The structure type name is not always explicit in the `sizeof`.

Example: hw/xen/xen_devconfig.c

```
    struct xs_dirs *d;

  - d = g_malloc(sizeof(*d));
  + d = g_new(xs_dirs, 1);
```

Sizeof expression

Semantic patch:

```
@@
identifier i;
struct i x;
@@

- g_malloc(sizeof(x))
+ g_new(i,1)
```

Coccinelle and type information:

- Coccinelle infers types in source code, when type information is available.
- The declaration of an expression metavariable can indicate a required type.
- This type can be used in matches and transformations.

Some examples

Example: hw/net/vhost_net.c

```
- struct vhost_net *net = g_malloc(sizeof *net);
+ struct vhost_net *net = g_new(vhost_net, 1);
```

Example: hw/i386/acpi-build.c

```
  CrsRangeEntry *entry;

- entry = g_malloc(sizeof(*entry));
+ entry = g_new(CrsRangeEntry, 1);
```

Some examples

Example: hw/net/vhost_net.c

- Depends on the paren isomorphism.

```
- struct vhost_net *net = g_malloc(sizeof *net);
+ struct vhost_net *net = g_new(vhost_net, 1);
```

Example: hw/i386/acpi-build.c

```
  CrsRangeEntry *entry;

- entry = g_malloc(sizeof(*entry));
+ entry = g_new(CrsRangeEntry, 1);
```

Some examples

Example: hw/net/vhost_net.c

- Depends on the paren isomorphism.

```
- struct vhost_net *net = g_malloc(sizeof *net);
+ struct vhost_net *net = g_new(vhost_net, 1);
```

Example: hw/i386/acpi-build.c

- Coccinelle finds information in the typedef.

```
  CrsRangeEntry *entry;

- entry = g_malloc(sizeof(*entry));
+ entry = g_new(CrsRangeEntry, 1);
```

Limitations of type inference

Observations:

- The rule updates 16 occurrences.
- There are 68 other occurrences with `sizeof(e)` for some expression `e`.
- We lack type information.

Getting more type information

- Typedefs are typically in header files.
- Default strategy: include .h files named like the .c file.
- Other options:

Arguments	Updated instances	Overlooked instances
	16	68
–all-includes -I qemu -I qemu/include	20	64
–recursive-includes -I qemu -I qemu/include	42	42

Some cases that are overlooked

block/commit.c:

```
typedef struct {
    int ret;
} CommitCompleteData;


CommitCompleteData *data;
...
data = g_malloc(sizeof(*data));
```

Some cases that are overlooked

block/commit.c:

```
typedef struct {
    int ret;
} CommitCompleteData;
```

```
CommitCompleteData *data;
...
data = g_malloc(sizeof(*data));
```

Change the source code to add a name for the structure type.

Some cases that are overlooked

block/qcow2-refcount.c:

```
typedef struct Qcow2DiscardRegion {
    BlockDriverState *bs;
    uint64_t offset;
    uint64_t bytes;
    QTAILQ_ENTRY(Qcow2DiscardRegion) next;
} Qcow2DiscardRegion;

Qcow2DiscardRegion *d, *p, *next;
...
d = g_malloc(sizeof(*d));
```

Some cases that are overlooked

block/qcow2-refcount.c:

```
typedef struct Qcow2DiscardRegion {
    BlockDriverState *bs;
    uint64_t offset;
    uint64_t bytes;
    QTAILQ_ENTRY(Qcow2DiscardRegion) next;
} Qcow2DiscardRegion;

Qcow2DiscardRegion *d, *p, *next;
...
d = g_malloc(sizeof(*d));
```

- Add `#define QTAILQ_ENTRY(x) x` to standard-qemu.h
- Spatch option: `--macro-file standard-qemu.h`
- 5 more occurrences get transformed

Some cases that are overlooked

ui/vnc.c

```
VncServerInfo *info;
...
info = g_malloc(sizeof(*info));

qapi-schema.json:
{ 'struct': 'VncServerInfo',
  'base': 'VncBasicInfo',
  'data': { '*auth': 'str' } }
```

Some cases that are overlooked

ui/vnc.c

```
VncServerInfo *info;
...
info = g_malloc(sizeof(*info));

qapi-schema.json:
{ 'struct': 'VncServerInfo',
  'base': 'VncBasicInfo',
  'data': { '*auth': 'str' } }
```

Can't help much with this issue...

Sizeof typedef

Our semantic patch:

```
@@
identifier s;
@@

- g_malloc0(sizeof(struct s))
+ g_new0(s, 1)
```

This doesn't match code like:

```
ALSAConf *conf = g_malloc(sizeof(ALSAConf));
```

Sizeof typedef

Issues:

- Need to find the typedef.
- Then find the structure type name.

Sizeof typedef

Issues:

- Need to find the typedef.
- Then find the structure type name.

Problem:

- Coccinelle matches only one top-level thing at a time.
- Typedef will be separate from the use of sizeof.

Sizeof typedef

Solution: Named rules and inheritance.

Match typedef:

```
@@
identifier i;
type t;
@@
typedef \( struct i \| struct i { ... } \) t;
```

Sizeof typedef

Solution: Named rules and inheritance.

Match typedef:

```
@nm@
identifier i;
type t;
@@
typedef \( struct i \| struct i { ... } \) t;
```

Sizeof typedef

Solution: Named rules and inheritance.

Match typedef:

```
@nm@
identifier i;
type t;
@@
typedef \( struct i \| struct i { ... } \) t;
```

Match sizeof:

```
@@
type nm.t;
identifier nm.i;
@@
- g_malloc(sizeof(t))
+ g_new(i,1)
```

Exercise 13

Another bite-sized task is to transform exit functions so that they have return type void.

We consider an exit function to be a function that is stored in the exit field of a structure:

```
    k->exit = esp_pci_scsi_uninit;
```

- Write a semantic patch to find initializations of an `exit` field. **Hint:** use `*`, as there is nothing to transform.
- Test your semantic patch on the `hw` directory.

Exercise 14

Continuing with exit functions, write a semantic patch that finds the definition of an exit function.

Hints:

- Write a semantic patch rule that matches an arbitrary function definition.
- Use inheritance to force this rule to match a function whose name was identified in the previous exercise.
- Remove the * from the rule for the previous exercise and put it on the function definition pattern.

Exercise 15

Continuing with exit functions, write a semantic patch that transforms an exit function into a function that has a void return type.

Hints:

- Replace the returns of a value in the body of the function by a `return;`
- Change the return type of the function to `void`

Exercise 16

To complete the transformation of exit functions, write a semantic patch to:

- Find any explicit call to an exit function and remove any use of its return value.
- Find any structure declaration that contains an `exit` field and change its return type.
- Find any call to the function stored in an `exit` field and remove any use of its return value.

NULL tests on g_malloc values are not needed

A plausible implementation:

```
@@
expression x;
statement S;
@@

  x = \(g_malloc\|g_malloc0\|g_new\|g_new0\)(...);
- if (x == NULL) S
```

Not general enough for all malloc cases:

```
datbuf = malloc(mtd->writesize + mtd->oobsize);
oobbuf = malloc(mtd->oobsize);
if (!datbuf || !oobbuf) {
  puts("No memory for page buffer\n");
  return 1;
}
```

NULL tests on `g_malloc` values are not needed

A more general implementation (search only)

```
@@
expression x, e;
statement S1,S2;
@@

  x = \(g_malloc\|g_malloc0\|g_new\|g_new0\)(...);
  ... when != x = e
* if (x == NULL || ...)
  S1 else S2
```

Isomorphisms allow:

- `|| ...` to be absent.
- `else S2` to be absent.
- `==` to be `!=`.

Results

A good result:

```
datbuf = malloc(mtd->writesize + mtd->oobsize);
oobbuf = malloc(mtd->oobsize);
if (!datbuf || !oobbuf) {
  puts("No memory for page buffer\n");
  return 1;
}
```

Another good result:

```
da8xx_fb_info = malloc(size);
debug("da8xx_fb_info at %x\n", (unsigned int)da8xx_fb_info);
if (!da8xx_fb_info) {
  printf("Memory allocation failed for fb_info\n");
  return NULL;
}
```

Results

May lead to false positives:

```
if (hfsp_vh->start_file.total_blocks != 0) {
  volume->boot_file = malloc(sizeof(hfs_fork_t));
  ...
} else {
  boot_id = hfsp_vh->finder_info[0];
}
...
if (volume->boot_file != NULL) {
  HFS_DPRINTF("Boot file:\n");
  hfs_dump_fork(volume->boot_file);
}
```

Results

May lead to false positives:

```
if (hfsp_vh->start_file.total_blocks != 0) {
  volume->boot_file = malloc(sizeof(hfs_fork_t));
  ...
} else {
  boot_id = hfsp_vh->finder_info[0];
}
...
if (volume->boot_file != NULL) {
  HFS_DPRINTF("Boot file:\n");
  hfs_dump_fork(volume->boot_file);
}
```

Problem: How to ensure that the NULL test is only reachable from the malloc?

Checking reachability

Two concepts:

- The NULL test is reachable from the malloc.
- The NULL test is reachable without performing the malloc.

Thus, two rules:

```
@r@
expression x, e; statement S1, S2;
@@
x = malloc(...);
... when != x = e
if (x == NULL || ...) S1 else S2

@@
expression r.x; statement r.S1, r.S2;
@@
... when != x = malloc(...);
if (x == NULL || ...) S1 else S2
```

Checking reachability

Two concepts:

- The NULL test is reachable from the malloc.
- The NULL test is reachable without performing the malloc.

Thus, two rules:

```
@r@
expression x, e; statement S1, S2;
@@
x = malloc(...);
... when != x = e
if (x == NULL || ...) S1 else S2

@@
expression r.x; statement r.S1, r.S2;
@@
... when != x = malloc(...);
if (x == NULL || ...) S1 else S2
```

How do we know that the NULL tests are the same?

Position variables

Position metavariables can be used to store the position of any token, for later matching or printing.

```
@r@
expression x, e; statement S1, S2; position p;
@@
x = malloc(...);
... when != x = e
if@p (x == NULL || ...) S1 else S2

@@
expression r.x; statement r.S1, r.S2; position r.p;
@@
... when != x = malloc(...);
if@p (x == NULL || ...)
S1 else S2
```

Completing the semantic patch: failure implies success

```
@r@
expression x, e; statement S1, S2; position p;
@@
x = malloc(...);
... when != x = e
if@p (x == NULL || ...) S1 else S2

@s@
expression r.x; statement r.S1, r.S2; position r.p;
@@
... when != x = malloc(...);
if@p (x == NULL || ...)
S1 else S2

@depends on !s@
expression r.x; statement r.S1, r.S2; position r.p;
@@
  x = malloc(...);
  ...
* if@p (x == NULL || ...) S1 else S2
```

Exercise 17

When searching for things, rather than transforming them, it may be useful to generate the output in a variety of formats. This can be done using the interface to python (ocaml is also available). Position variables are useful in this context, because they provide the file name and line number of various program elements.

Consider the following semantic patch, presented earlier.

```
@@
identifier x,y; identifier f; statement S;
@@
  x = f(...);
(
  if (x < 0) S
|
  if (
-     y
+     x
      < 0) S
)
```

Exercise 17, contd.

The following python code is intended to print the file name and line numbers of the assignment and erroneous test, respectively:

```
@script:python@
p1 << r.p1; // inherit a metavariable p1 from rule r
p2 << r.p2; // inherit a metavariable p2 from rule r
@@
print p1[0].file, p1[0].line, p2[0].line
```

Create a semantic patch consisting of the original patch rule shown on the previous page followed by the above python code. Rewrite the patch rule to:

- Give the rule the name, `r`.
- Remove the transformation.
- Add position variables `p1` and `p2`.
- Attach the position variables to relevant code.
- Test the result on roms/u-boot.

Exercise 18

We have seen that * can be used to highlight items of interest. Repeat the previous exercise, this time without using python, but instead annotate the original code pattern with * rather than performing transformations.

How is the result different than the result produced when using python?

Exercise 19

Recall the elimination of uses of the QError macros QERR_ covered in Exercise 6. Write a semantic patch involving the use of python to eliminate uses of QError macros and replace occurrences of %s by the constant string argument, when there is one.

An example of the desired transformation is as follows:

```
- error_setg(errp, QERR_INVALID_PARAMETER, "speed");
+ error_setg(errp, "Invalid parameter 'speed'");
```

Hint: A metavariable of type `constant char[]` matches an explicit string constant.

Putting it all together

- Convert `malloc`/`free` to `g_malloc`/`g_free`.
- Convert `g_malloc` to `g_malloc0`.
- Convert `g_malloc`/`g_malloc0` to `g_new`/`g_new0`.
- Drop NULL tests on the results of these functions.

Feature summary

- `when` annotations.
- Nests.
- Isomorphisms.
- Reasoning about types.
- Named rules and metavariable inheritance.
- Position variables.
- Scripting using python.

Other opportunities in Qemu

Bite sized tasks, ongoing transitions:

- QemuMutex/QemuCond → CompatGMutex/CompatGCond.
- Include SDState by value instead of allocating it in sd_init.
- get_ticks_per_sec() → NSEC_PER_SEC.
- qemu_system_reset_request() → watchdog_perform_action().
- fprintf(stderr,...) → error_report()
- etc.

Other issues:

- Missing frees, missing unlocks.
- Non-use of standard macros, helper functions.
- etc.

Conclusion

Coccinelle:

- Code-like matching and transformation language.
- Flexibility via a small set of features (isomorphisms, types, 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
- 21 Qemu patches mention Coccinelle
- Many more opportunities seem to be present in Qemu!

http://coccinelle.lip6.fr

http://btrlinux.fr

http://coccinellery.org