Coccinelle Introduction

Michael Stefaniuc
mstefani@winehq.org
Wine Project
13th October 2019

Disclaimer

- Wine is **just** a hobby for me
- I'm speaking for myself and not for my employer
- Trip is not sponsored

Coccinelle

“Coccinelle is a program matching and transformation engine which provides the language SmPL (Semantic Patch Language) for specifying desired matches and transformations in C code.“ (http://coccinelle.lip6.fr/)

Coccinelle History

- Academic research project
  - Collaboration between DIKU and INRIA
  - First published paper 2005
  - Initially known as Tarantula (name change ~2005)
  - Focus on “Collateral Evolution” in the Linux kernel
- Jan 2009 LWN article “Semantic patching with Coccinelle” by Valerie Henson
- In the Linux kernel tree since 2010 (checker / cocci scripts)

Coccinelle Project

- Actively maintained
- Written in OCaml
- Current version: 1.0.8 (25th Sept 2019)
- Maintainer: Julia L. Lawall
- Website: http://coccinelle.lip6.fr/
- Source code: https://github.com/coccinelle
- Mailing list: cocci@systeme.lip6.fr

Coccinelle Wine History

- ~2009 - First steps: long ==> LONG for 64-bit support
- 2010 - First credit in Wine git
- 2010 – Initial commit of my published cocci scripts for Wine
- Major work:
  - COM cleanup
  - ARRAY_SIZE
  - Majority of my un-credited janitorial work
  - d3dx9 effect cleanup

Glossary

- Project: coccinelle
- Script language: SmPL (Semantic Patch Language)
- Script file extension: .cocci
- Executable: spatch (not coccinelle)

Golden Rule

Coccinelle knows C!

My First SmPL

```
@@
typedef LONG;
@@
- long
+ LONG
```

- SmPL looks like
  - Unified diff
  - With C declarations
- Header enclosed in @@ @@
  - Declares metavariables
  - Name and control info between first @@
- Body
  - Context
  - Add / Remove lines

ARRAY_SIZE.cocci

```
@ r @
type T;
T[] E;
position p;
@@
(
- (sizeof(E@p)/sizeof(E[...]))
+ ARRAY_SIZE(E)
|
- (sizeof(E@p)/sizeof(*E))
+ ARRAY_SIZE(E)
|
- (sizeof(E@p)/sizeof(T))
+ ARRAY_SIZE(E)
)
```

- Header
  - E == expression of type array of T
  - position == records specific position inside the code
- Body
  - Disjunction
  - “…” operator == anything

comma.cocci

```
@@
expression E1, E2;
statement S;
@@
  S
  E1
- ,
+ ;
  E2;
```

- Also valid body

```
  S
- E1, E2;
+ E1; E2;
```

- Smallest change possible, avoids
  - code reformatting
  - whitespace changes
  - loosing comments
- Whole SmPL body needs to be a full and valid C construct!
  Before and after!

merge.cocci

```
@base@
identifier virtual.func;
statement list body;
type T;
@@
- T func(...) { body }

@@
identifier virtual.func;
statement list base.body;
@@
- return func(...);
+ body
```

- Using metavariable from another rule: `rulename.metavariable`
- Rule “virtual” specified on command line: `spatch -D func=foo`
- All inherited variables need to be bound for a rule to be executed
- Runs once for each set of bound variables
- list metavariables

wstr.cocci

```
@r@
identifier lvar;
initializer list chs;
@@
 WCHAR lvar[]@p = { chs, \('\0'\|0\) };

@script:python u@
lvar << r.lvar;
chs << r.chs;
wstr;
@@
coccinelle.wstr = 'L"' + "".join(map(lambda x: x[1:-1], chs)) + '"'

@@
identifier r.lvar, u.wstr;
initializer list r.chs;
@@
 WCHAR lvar[]@p =
-                { chs, \('\0'\|0\) }
+                wstr
                 ;
```

- Script rules:
  - OCaml
  - Python
- `@initialize:python@` Runs once before any other rule
- `@finalize:python@` Runs once after all matching was done

redundant_null_check.cocci

```
@@
expression E;
type T;
identifier fn = {CoTaskMemFree, free, Free,
GdipFree, HeapFree, heap_free, I_RpcFree, msi_free,
MSVCRT_free, MyFree, RtlFreeHeap, SysFreeString};
@@
(
- if (E != NULL)
      fn(..., (T)E);
|
- if (E != NULL)
- {
      fn(..., (T)E);
?     E = NULL;
- }
)
```

- Constraints on metavariables
- Isomorphism:
  Automatic transformation of SmPL

```
(
-if -(-E -!= -NULL-)
|
-if -(-E-)
|
-if -(-NULL -!= -E-)
)
(
fn(..., (T )E);
|
fn(..., E);
)
```

… Dots

- `'...'` Matches the shortest path between stuff before/after the dots
- `'<...  ...>'` Matches 0 or more times the stuff between the ellipses
- `'<+...  ...+>'` Matches 1 or more times the stuff between the ellipses on some path
- Constraints:

```
... when any
... when exists
... when strict
... when != x
```

Running spatch

- As simple as
```
spatch foo.cocci bar.c
spatch foo.cocci directory/
```
- Multiple files as one compilation unit
```
spatch foo.cocci bar.c barf.c foobar.c
spatch foo.cocci directory/*.c
```
- Checking the cocci script
```
spatch --parse-cocci foo.cocci
```

Running spatch for Wine

- Default coccinelle macro file for the Linux kernel
- Heavy C parse issues due to macros

```
spatch --parse-c dlls/mshtml/tests/style.c
nb good = 1570,  nb passed = 35 =========> 0.93% passed
nb good = 1570,  nb bad = 2158 =========> 42.65% good or
passed
```

- Solution: Use macro file for Wine

```
spatch --macro-file-builtins macros dlls/mshtml/tests/style.c
nb good = 3728,  nb passed = 12 =========> 0.32% passed
nb good = 3728,  nb bad = 0 =========> 100.00% good or passed
```

- Use my ‘`coccicheck`’ wrapper around spatch

Dealing with Types in Wine

- Copious use of ‘typedef’
- Dealing with multiple identical types
  - Constraints
    ```
    type lpjunk = {LPJUNK, PJUNK, IJunk*};
    ```
  - Get rid of them
    ```
    @@ typedef LPJUNK, PJUNK, IJunk; @@
    (
    - LPJUNK
    + IJunk *
    |
    - PJUNK
    + IJunk *
    )
    ```

Include Files

- Default is to include local headers (`--local-includes`)
- Avoid using include files if possible (`--no-includes`)
  Performance!
- When processing a directory include files are skipped.
  Include them with (--include-headers)

Resources

- Better tutorials:
  - http://coccinelle.lip6.fr/documentation.php
  - http://coccinelle.lip6.fr/papers.php
- Examples
  - Coccinelle git tree in `demos/`
  - Linux kernel git tree in `scripts/coccinelle/`
- My Wine coccinelle scripts
  https://github.com/mstefani/coccinelle-wine.git