Introduction to Coccinelle

Semantic patches for automated code modification and more...
What is coccinelle

• Definition: “program matching and transformation engine which provides the language SmPL (Semantic Patch Language) for specifying desired matches and transformations in C code”

• Target:
  • Make structural modification on a large codebase
  • Detect programmatic errors
  • ...

• Used in Linux kernel
Semantic patches

- Initial version of the software was using a language only known by some university guys
- They decide to switch to a patch like format
- Example

```plaintext
@rule1@
identifier p;
identifier func;
@@
func(...) {
  ...
  Packet p;
  ...
  - & (p)
  + p
  }
```
A powerful system (1/2)

• It understands C

• Here's a semantic patch:

```c
@rule2@
identifier p;
identifier func;
@@
func(...) {
  <...
  - Packet p;
  + Packet *p = SCMalloc(SIZE_OF_PACKET);
  + if (p == NULL) return 0;
  ...
  + SCFree(p);
  return ....;
  ...>
}
```
Here's the result

the SCFree call has been put before all return:

```c
-    ret = Unified2Alert(&tv, &p, data, &pq, NULL);
-    if(ret == TM_ECODE_FAILED)
  +    }
+    ret = Unified2Alert(&tv, p, data, &pq, NULL);
+    if(ret == TM_ECODE_FAILED) {
+      SCFree(p);
  return 0;
+    }
+    ret = Unified2AlertThreadDeinit(&tv, data);
-    if(ret == -1)
+    if(ret == -1) {
+       SCFree(p);
  return 0;
+    }
```
Difficult to master

• Syntax can be really tricky
• “special” case needs to be handle manually
  • Multiple substitutions “<... ...>”
  • Multiple line adding “++”
• The semantic patches are chained together
  • Dependancy system
Interesting features

- Python integration
  - You can use python inside the semantic patches
  - To print result or do some advanced checking
- Regular expression usage
  - To match identifier with specific name
  - ...

...
Alternative usage

- Testing and unittesting
  - Use like this in Suricata
  - Respect of API
  - Respect of code convention
- Code matching
  - Coccigrep
Coccinelle in testing

- Do semantic patches using match
- Display output

```c
if (zeroed)
    typedef Packet;
    typedef uint8_t;
    Packet *p;
    position p1;
    @

    memset(p@p1, 0, ...);

if (isset)
    Packet *p;
    position zeroed.p1;
    @

    memset(p@p1, 0, ...);
    ... when != p
    p->pkt

@script:python depends on !isset@
    pl <= zeroed.p1;
    @

    print "Packet zeroed at %s:%s but pkt field is not set afterward." % (pl[0].file, pl[0].line)
import sys
    sys.exit(1)
```
Coccigrep (1/2)

- Semantic grep using coccinelle
- Command line tool

```bash
eric@tiger:~/git/oisf/src (af_packet_v1) $ coccigrep -t Packet -c -a datalink -o set source*c
  source-af-packet.c: l.313 -0, l.313 +0, Packet *p
    p->datalink = ptv->datalink;
  source-erf-dag.c: l.525 -0, l.525 +0, Packet *p
    p->datalink = LINKTYPE ETHERNET;
  source-erf-file.c: l.138 -0, l.138 +0, Packet *p
    p->datalink = LINKTYPE ETHERNET;
  source-ipfw.c: l.256 -0, l.256 +0, Packet *p
    p->datalink = ptv->datalink;
  source-nfq.c: l.323 -0, l.323 +0, Packet *p
    p->datalink = DLT_RAW;
  source-pcap.c: l.169 -0, l.169 +0, Packet *p
    p->datalink = ptv->datalink;
  source-pcap.c: l.268 -0, l.268 +0, Packet *p
    p->datalink = ptv->datalink;
  source-pcap-file.c: l.126 -0, l.126 +0, Packet *p
    p->datalink = pcap_g.datalink;
  source-pfring.c: l.194 -0, l.194 +0, Packet *p
    p->datalink = LINKTYPE ETHERNET;
```
Coccigrep (2/2)

- Tests:
  - Set: structure attribut is set
  - Func: structure is used as parameter of a function
  - Used: structure is used
  - Test: attribut of the structure is used in test
  - Derref: attribut of a structure is used

- Integration in editor
  - Vim currently supported
  - Who wants emacs support?
More information

- Project website: http://coccinelle.lip6.fr/
- Great support through the mailing list:
  - http://coccinelle.lip6.fr/contact.php
  - Julia Lawall is excellent
- Coccinelle for the newbie:
  http://home.regit.org/technical-articles/coccinelle-for-the-newbie/
- Coccigrep:
  http://home.regit.org/software/coccigrep/