Automatic Generation of Counter-Measure Calling Contexts: the CURSOR Method

Dassault Aviation
Dillon Pariente

June 20th, 2016 – Paris, La Maison de La Recherche

Outline

- Context
- Objectives and Case Study on Security
- CURSOR Method Principles
- Demo
- Conclusion & Perspectives
- Q&A

Context

FP7 - STANCE Project (2012-2016)

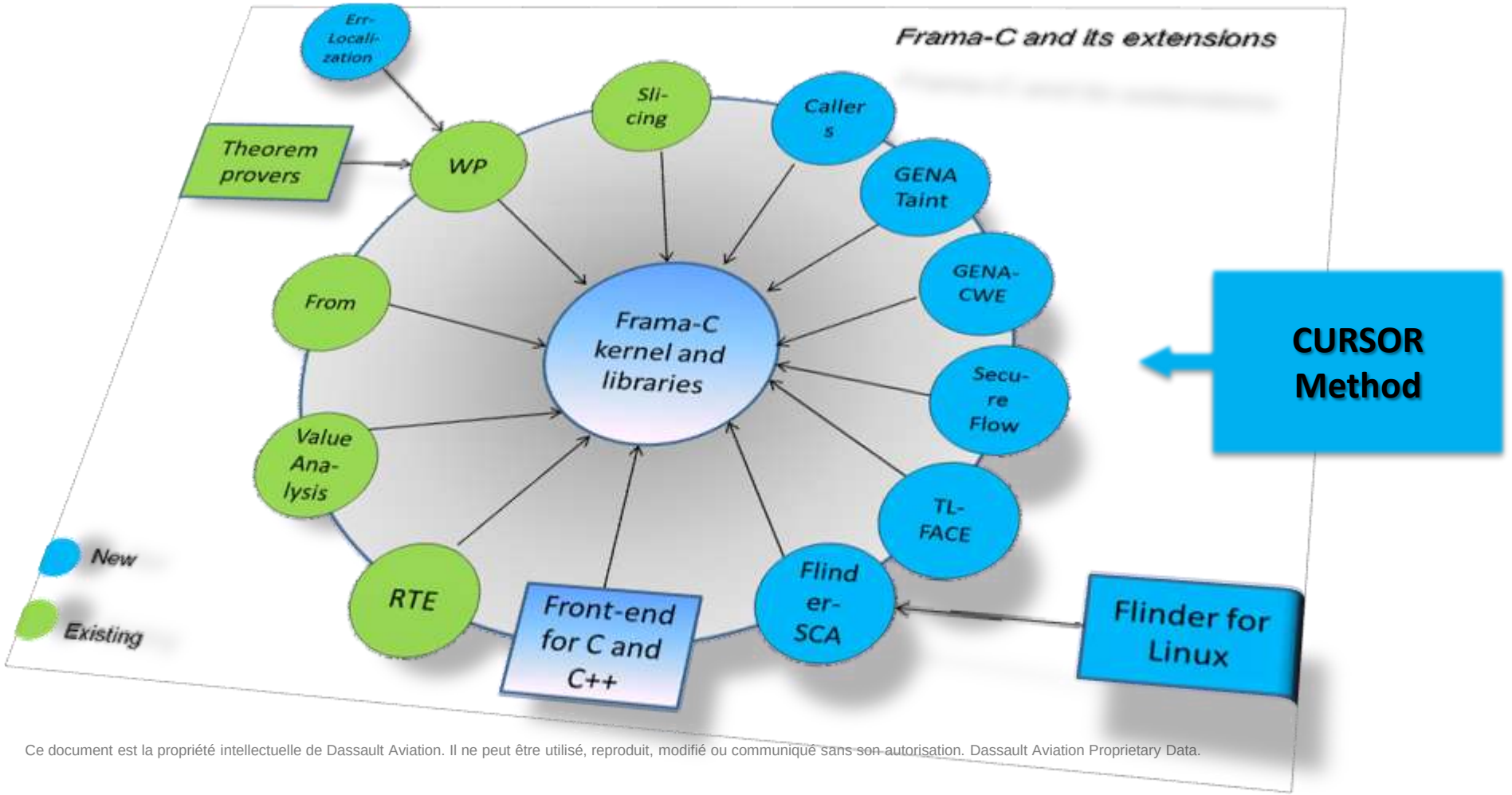
Partners: CEA-List (F, Leader), Trusted-Labs (F), Thales CS (F), KULeuven (B), Infineon (G), SearchLab (H), Arttic (F), TU Graz (A), Fraunhofer (G), Dassault Aviation (F)

Main purposes: extending Frama-C for Security, C++ front-end, …

http://stance-project.com (links to the SW, publications, reports)

Context (cont'd)

FP7 STANCE's outcomes: partial outline

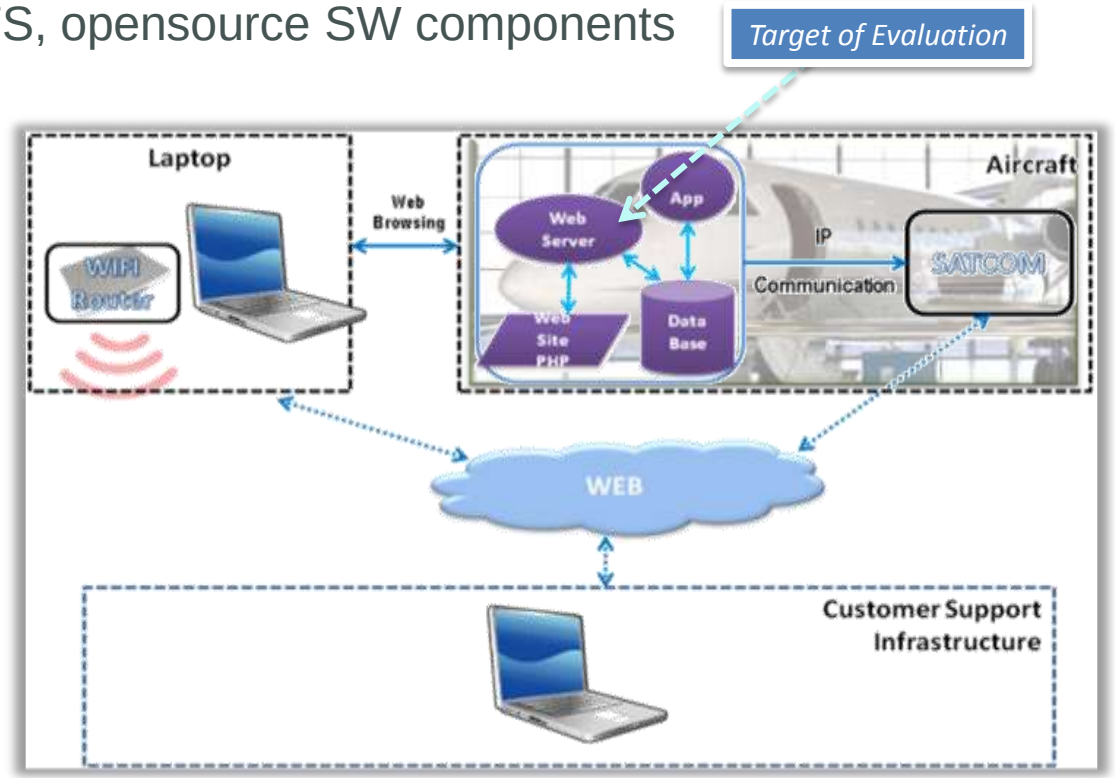


Objectives / Case Study

- **Aircraft e-Maintenance** (*on-going dev.*)

  Embedded web application with COTS, opensource SW components

  Apache, OpenSSL, ProgreSQL, …



⇒ How to detect CWE by FM on some critical components?

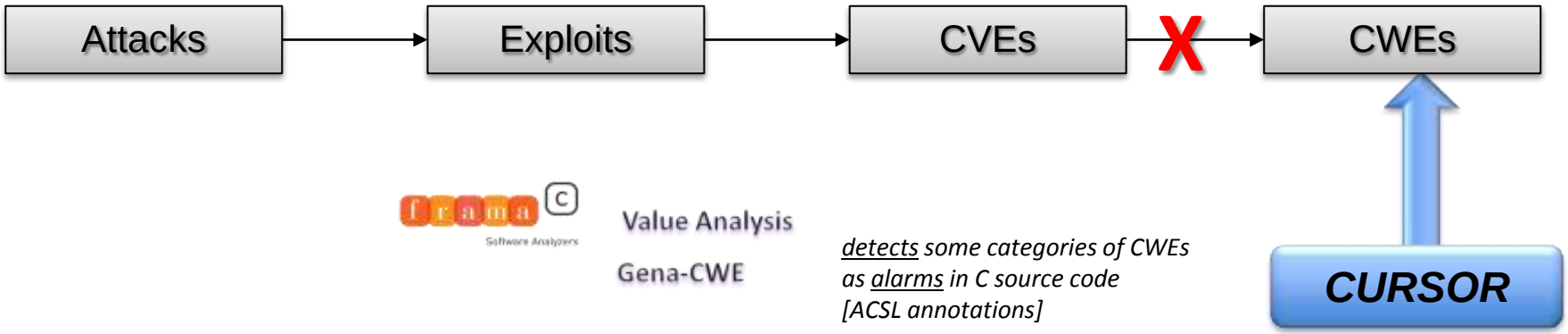
⇒ How to strengthen the code? (cost-)efficiently? ...

CURSOR method

***C**omponent or **U**nit **R**obustness for **S**ecurity **O**bjectives and **R**equirements*

DASSAULT AVIATION

How to break this causality chain?



detects some categories of CWEs as alarms in C source code [ACSL annotations]

automatically translates (most of) these alarms as executable code [counter-measure calling contexts]

RAC approach
Runtime Assertion Checking

requires to implement these counter-measures

defensive programming

log *stop service* *...* *add IPS rules*

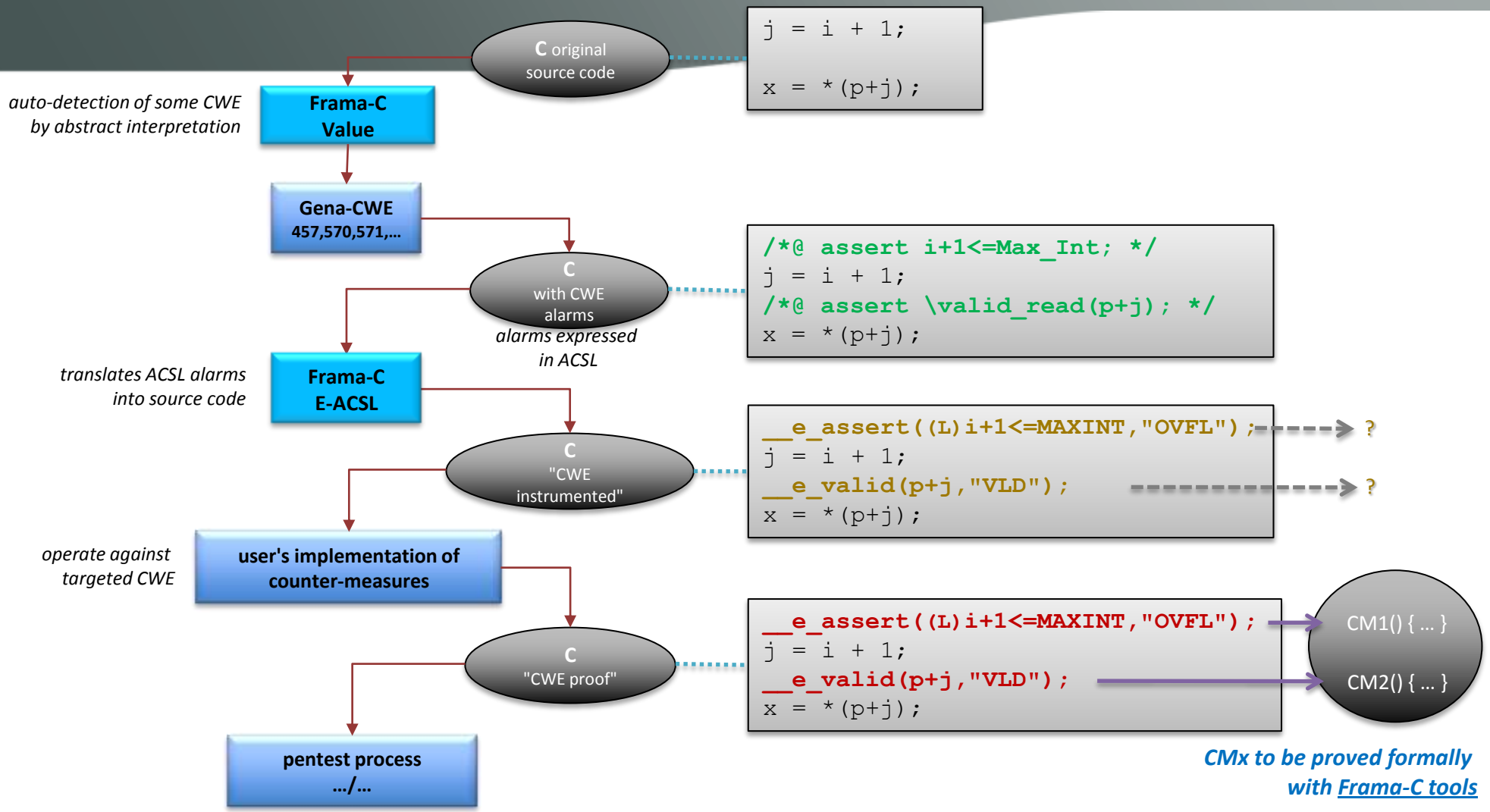
Advantages:

- pretty **automatic** approach
- contributes to **Security Function** implementation
- impacts **attack surface** response
- **no expertise** required in Formal Method

may rely on

CURSOR method

Component or Unit Robustness for Security Objectives and Requirements



CURSOR – application to Apache

(server/util library)

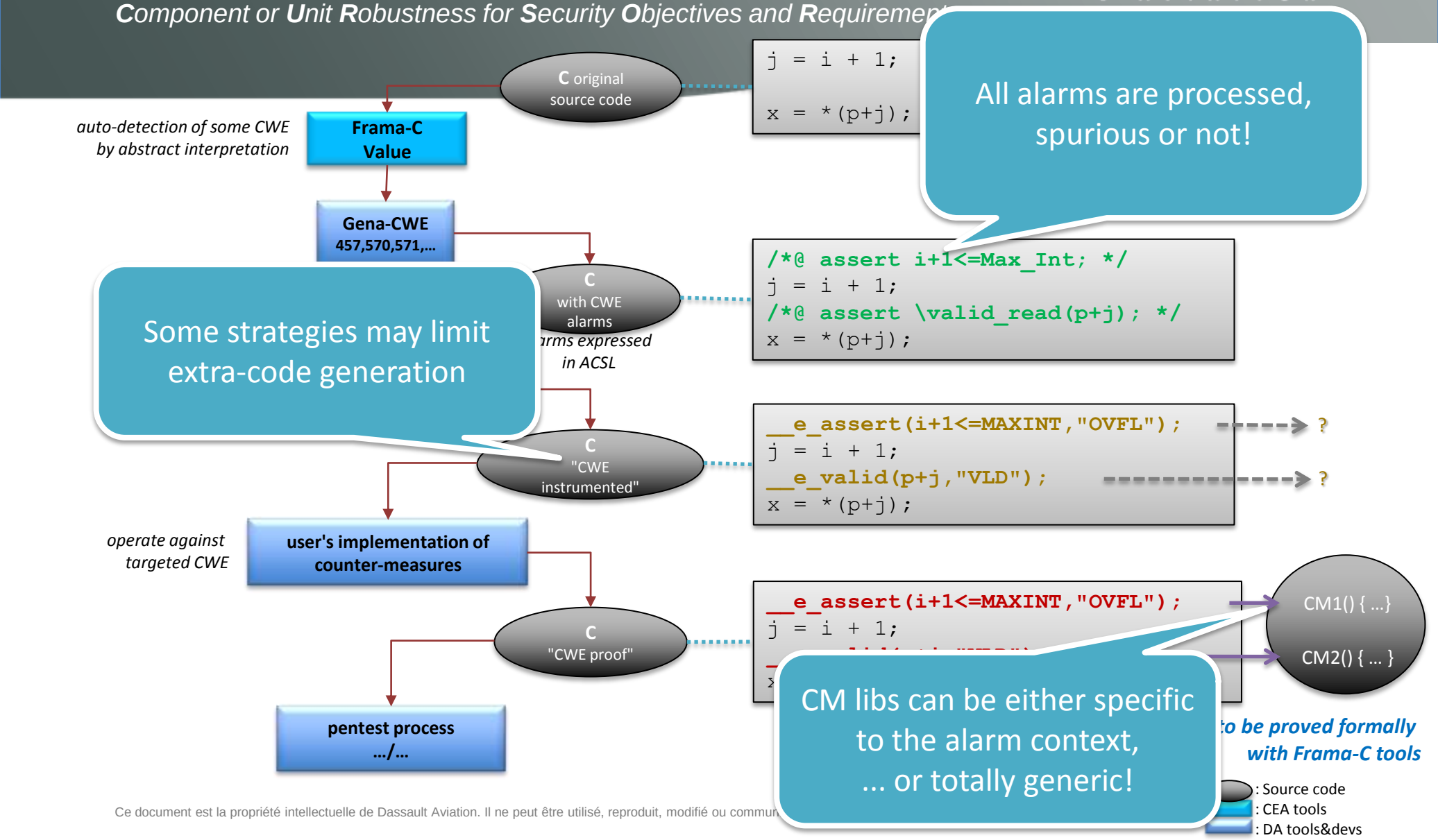
```
          int __e_acsl_valid_read_3;
          __e_acsl_valid_read_3 = __valid_read((void *)(y +
2),sizeof(char));
          e_acsl_assert(__e_acsl_valid_read_3,(char *)"Assertion",
                        (char *)"unescape_url",
                        (char *)"Value: mem_access:
\\valid_read(y+2)",7829);
        }
        loc_tmp_6 = __e_acsl_isxdigit((int)((unsigned char)*(y +
2)));
        if (loc_tmp_6) {
          char decoded;
          decoded = x2c((char const *)(y + 1));
          if ((int)decoded == '\000') goto _LOR;
          else
            if (forbid) {
              char *loc_tmp_4;
              loc_tmp_4 = __e_acsl_strchr(forbid,(int)decoded);
              /*@ assert Value: ptr_comparison:
                \pointer_comparable((void *)0, (void *)loc_tmp_4);
               */
              ;
              if (loc_tmp_4) {
                _LOR: badpath = 1;
                /*@ assert Value: mem_access: \valid(x); */
              {
                int __e_acsl_initialized_3;
                int __e_acsl_and_3;
                __e_acsl_initialized_3 = __initialized((void *)(& x),
                                                       sizeof(char *));
                    if (__e_acsl_initialized_3) {
                      int __e_acsl_valid_2;
                      __e_acsl_valid_2 = __valid((void
*)x,sizeof(char));
                      __e_acsl_and_3 = __e_acsl_valid_2;
                    }
                    else __e_acsl_and_3 = 0;
                    e_acsl_assert(__e_acsl_and_3,(char *)"Assertion",
                                  (char *)"unescape_url",
                                  (char *)"Value: mem_access:
\\valid(x)",7846);
                  }
                  *x = decoded;
                  y += 2;
                }
                else goto _LAND_0;
              }
              else {
                _LAND_0: ;
                if (reserved) {
                  char *loc_tmp_3;
                  loc_tmp_3 = __e_acsl_strchr(reserved,(int)decoded);
                  /*@ assert Value: ptr_comparison:
                    \pointer_comparable((void *)0, (void *)loc_tmp_3);
                   */
                  ;
                  if (loc_tmp_3) {
                    char *loc_tmp;
```

DEMO

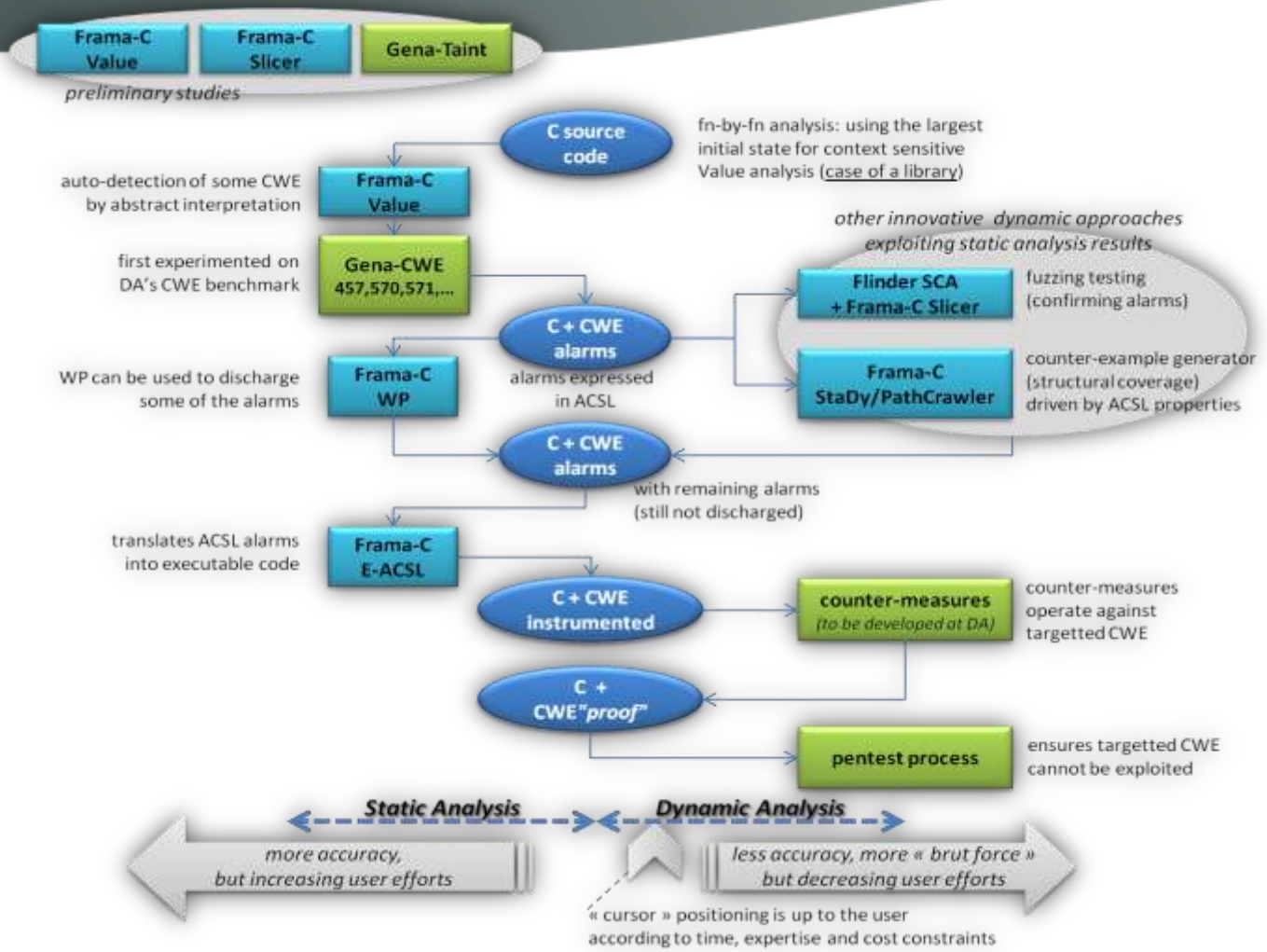
```
                  char *loc_tmp_0;
                  char *loc_tmp_1;
                  char *loc_tmp_2;
                  loc_tmp = x;
                  x ++;
                  loc_tmp_0 = y;
                  y ++;
                  /*@ assert Value: mem_access: \valid(loc_tmp); */
                  {
                    int __e_acsl_initialized_4;
                    int __e_acsl_and_4;
                    __e_acsl_initialized_4 = __initialized((void *)(&
loc_tmp),

sizeof(char *));
                    if (__e_acsl_initialized_4) {
                      int __e_acsl_valid_3;
                      __e_acsl_valid_3 = __valid((void *)loc_tmp,
                                                 sizeof(char));
                      __e_acsl_and_4 = __e_acsl_valid_3;
                    }
                    else __e_acsl_and_4 = 0;
                    e_acsl_assert(__e_acsl_and_4,(char *)"Assertion",
                                  (char *)"unescape_url",
                                  (char *)"Value: mem_access:
\\valid(loc_tmp)",
                                  7872);
                  }
                  *loc_tmp = *loc_tmp_0;
                  loc_tmp_1 = x;
                  x ++;
                  loc_tmp_2 = y;
                  y ++;
                  /*@ assert Value: mem_access: \valid(loc_tmp_1); */
                  {
                    int __e_acsl_initialized_5;
                    int __e_acsl_and_5;
                    __e_acsl_initialized_5 = __initialized((void *)(&
loc_tmp_1),

sizeof(char *));
                    if (__e_acsl_initialized_5) {
                      int __e_acsl_valid_4;
                      __e_acsl_valid_4 = __valid((void *)loc_tmp_1,
                                                 sizeof(char));
                      __e_acsl_and_5 = __e_acsl_valid_4;
                    }
                    else __e_acsl_and_5 = 0;
                    e_acsl_assert(__e_acsl_and_5,(char *)"Assertion",
                                  (char *)"unescape_url",
                                  (char *)"Value: mem_access:
\\valid(loc_tmp_1)",
                                  7881);
                  }
                  *loc_tmp_1 = *loc_tmp_2;
                  /*@ assert Value: mem_access: \valid(x); */
                {
                  int __e_acsl_initialized_6;
```

CURSOR method

***C**omponent or **U**nit **R**obustness for **S**ecurity **O**bjectives and **R**equirement*



CURSOR – a more sophisticated process



Conclusion and Perspectives

- Simple and straightforward use of Frama-C plug-ins
- First experiments with CURSOR method are quite conclusive (R&T context)
- Still applicable to more critical/complex applications? (relevant counter-measures? …)

- How CURSOR may contribute to Security Evaluation process, Common Criteria certification, …?