FlexeLint

Kort introduktion til FlexeLint statisk source code analyse

Af Ivan Skytte Jørgensen

Agenda

- Orientering / hvad er Flexelint
- Simple eksempler
- Advarsler
- Undertrykkelse af advarsler
- Diverse og opsummering

Orientering / hvad er Flexelint

Hvad er FlexeLint?

- FlexeLint er et værktøj til statisk analyse af C/C++ source code
- Det kan finde fejl i programmer:
  - Tvivlsomme konstruktioner
  - Memory leaks
  - Brug af uinitialiseret data
  - Buffer overflows
  - Død code
  - Og meget, meget mere

Statisk versus dynamisk analyse

- Dynamisk analyse kører programmet
- Statisk analyse inspicerer kildekoden

Dynamisk analyse

- Kører programmet med nogle testdata
- Finder faktiske fejl
- Finder ikke fejl i de dele af programmet som ikke blev kørt
- Finder ikke fejl for inputværdier som ikke blev brugt
- Finder måske ikke tidsfølsomme fejl
- Kun så god som dine testdata

Værktøjer til dynamisk analyse

- OS (core dump)
- Valgrind
- Rational Purify
- Parasoft Insure++
- Intel Inspector

Statisk analyse

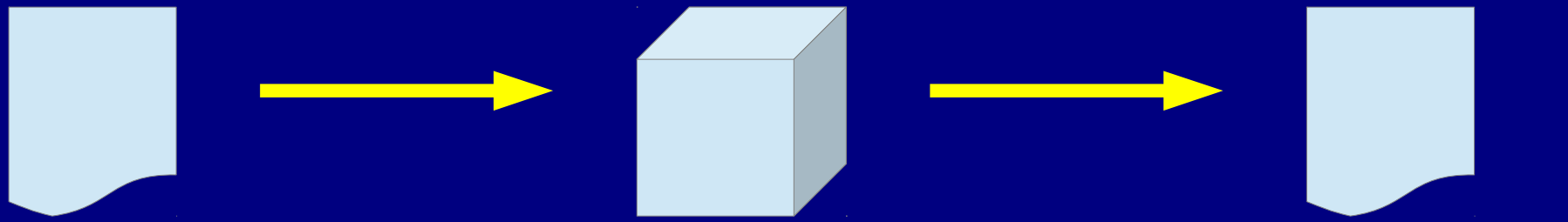
- Inspicerer kildekoden
- Finder potentielle fejl
- Kigger på alle logiske kørsels-”stier”
- Vil finde fejl som ikke er fejl
- Teoretisk et umuligt problem (f.eks. jvf. Halting Problem)

Værktøjer til statisk analyse

- Compiler warnings
- Gimpel FlexeLint/PCLint
- Fujitsu PGRelief
- Parasoft CodeWizard
- Abraxas CodeCheck
- HP CodeAdvisor
- Polyspace
- Coverity
- Klocwork

Forventninger (1)

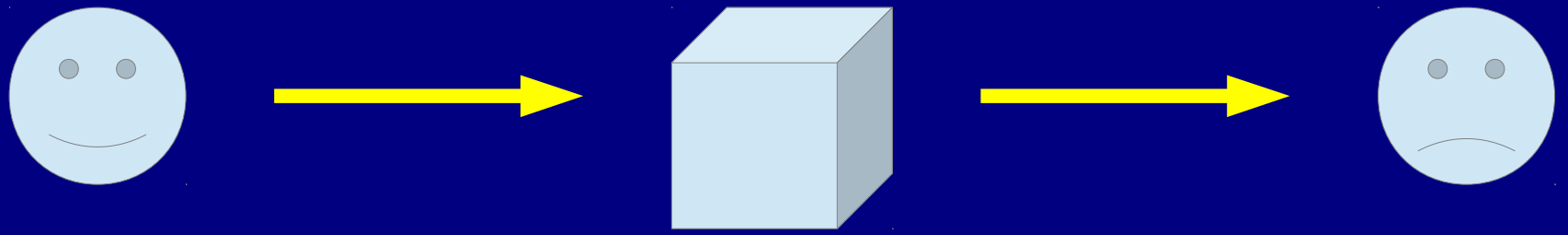
- Forkerte forventninger:



Fejlbehæftet program FlexeLint Fejlfrit program

Forventninger (2)

- Rigtige forventninger:



Glad udvikler

FlexeLint

Realistisk udvikler

Eksempler

- Jeg har fundet eller skrevet noget slamkode til lejligheden...

Eksempler (1a)

```
#include <string.h>

long& suspect_function(long magic, int x, int y) {
    long tmp;
    memcpy(&tmp,&x,sizeof(tmp));
    long result = magic/tmp;
    result /= y;
    return result;
}

void foo() {
    long l = suspect_function(100l,42,0);
}
```

Eksempler (1b)

```
        memcpy(&tmp,&x,sizeof(tmp));
t266.cc  5  Warning 420: Apparent access beyond array for function 'memcpy(void *,
const void *, unsigned long)', argument 3 (size=8) exceeds argument 2 (size=4)

      _
        return result;
t266.cc  8  Warning 1571: Returning an auto variable 'result' via a reference type

                          _
        long l = suspect_function(100l,42,0);
t266.cc  12  Warning 620: Suspicious constant (L or one?)

_
}
t266.cc  13  Warning 438: Last value assigned to variable 'l' (defined at line 12) not used

_
}
t266.cc  13  Warning 529: Symbol 'l' (line 13) not subsequently referenced

During Specific Walk:
  File t266.cc line 12: suspect_function(100, 42, 0) #1
t266.cc  7  Warning 414: Possible division by 0 [Reference: file t266.cc: line 12]
t266.cc  12  Info 831: Reference cited in prior message
```

Eksempler: præcision (1)

- FlexeLint genererer meget præcise advarsler, som kigger på sammenhængen
- I nedenstående eksempel kommer der kun en advarsel, hvis divisionsresten faktisk kunne blive brugt

```
void foo(int x, int y) {
    int    z1 = x/y;  //no warning
    double z2 = x/y;  //Warning 653: Possible loss of fraction
}
```

Eksempler: præcision (2)

- Warning 603 er for uinitialiserede variabler som overføres i en const* parameter

```
extern void foo1(const int *);
extern void foo2(int *);

void boo(void) {
    int x;
    int y=0;
    int z;
    foo1(&x); //warning 603: Symbol 'x' not initialized
    foo1(&y); //no warning
    foo2(&x); //no warning
    foo2(&y); //no warning
    foo2(&z); //no warning
    foo1(&z); //no warning
}
```

Advarsler fra Flexelint

- FlexeLint genererer over 900 forskellige advarsler
  - Syntaksfejl
  - Decideredе fejl
  - Obskure semantiske detaljer
  - “Dårlig stil”
  - MISRA (se evt. senere)
  - Brugerefterspurgte fejl siden 1984

Advarsler, uddrag: 432

432 Suspicious argument to malloc -- The following pattern was detected:

```
malloc( strlen(e+1) )
```

where e is some expression. This is suspicious because it closely resembles the commonly used pattern:

```
malloc( strlen(e)+1 )
```

If you really intended to use the first pattern then an equivalent expression that will not raise this error is:

```
malloc( strlen(e)-1 )
```

Advarsler, udrag: 1537

1537 const function returns pointer data member 'Symbol'
-- A const function is behaving suspiciously. It is returning a pointer data member (or equivalently a pointer to data that is pointed to by a data member). For example,

```
class X
{
    int *p;
    int *f() const { return p; }
};
```

Since f is supposedly const and since p is presumptively pointing to data that is logically part of class X we certainly have the potential for a security breach. Either return a pointer to const or remove the const modifier to the function. [12, Item 29].

Note, if a const function returns the address of a data member then a 605 (capability increase) is issued.

Advarsler, udrag: 960

Message 960 er MISRA warnings. Nogle kan give god mening at enable i normale programmer. F.eks.:

```
(Rule 42/12.10) Comma operator used outside of 'for' expression.
(Rule 14.10) No 'else' at end of 'if ... else if' chain.
(Rule 15.4) Boolean value in switch expression
```

Hvordan man lukker munden på FlexeLint

- Man kan undertrykke advarsler med høj præcision:
  - Globalt
  - Pr. fil
  - Pr. symbol
  - Pr. type
  - Pr. linje (i kildekoden)
  - Pr. {} blok (i kildekoden)
- Normalt vil man putte konfiguration og undertrykkelser i en separat fil, som man så specificerer som input til Flexelint

flexelint.lnt eksempel

```
-esym(765,module_descriptor)   //”extern could be made static”
                               //No, it couldn't

-esym(534,SHA1_Init,SHA1_Update,SHA1_Final)

//We don't really need to check return value
-esym(534,AAA::SimplifiedHookContext::setProperty_*)

//We often do "int hook_return = hrc_ok;"
-estring(641,AAA::HookReturnCode)    //Converting enum
                                     //'AAA::HookReturnCode' to
                                     //'xxxxx'
```

Undertryk advarsel for et symbol

- +esym/-esym enabler/disabler en advarsel for et symbol
- Eksempel:

```
//...is only referenced by its c'tor and d'tor
-esym(1788,file_lock)
//...return value is not used
-esym(534,printf)
```

- Wildcards er også understøttet:

```
-esym(534,et_faelles_prefix*)
```

Undertryk advarsel for en type

- Nogle gange er “etype” bedre
- Eksempel:

```
//..is only referenced by its c'tor and d'tor
-esym(1788,boost::lock_guard<*>)
```

Flere advarsler: -sem

- -sem() fortæller Flexelint om semantikken I funktioner som den ikke har kildekoden til
- Eksempel from logger.h:

```
void mesgHex(int error_code, const void* raw_data,
             size_t raw_len, const char* fmt, ...);
```

- flexelint.lnt:

```
-sem(mesgHex,2P>=3n,pod(2))
```

- Dette fortæller FlexeLint at *raw_data* skal have mindst *raw_len* bytes
- Og at *raw_data* ikke må indeholde et ikke-trivielt C++ object (non-POD)

Resourcer

- Online: www.gimpel-online.com/OnlineTesting.html
- Uddrag af manualen: www.gimpel.com/html/manual.pdf (man får kun hele manualen hvis man køber)
- www.gimpel.com
- http://www.gimpel.com/discussion.cfm
- Disse slides er tilgængelige på http://i1.dk/

Slut

Ekstra slides

Hvem bruger statisk analyse?

- Brancher hvor omkostningerne ved fejl kan måles objektivt (i hvert fald delvist)
  - Medical
  - Finansiel
  - Bilindustrien
  - Luftfart
  - Aerospace
  - Militær (forhåbentlig!)

Vil FlexeLint finde fejl i min kode?

- Hvis det er moden koden, og udviklerne har vedligeholdt det med omhu: Ikke ret mange.
- Så: Mest grænsetilfælde (“corner-cases”)
- Men normalt mere alvorlige problemer i ny kode

Andre C/C++ statisk analyse værktøjer

- Mange er blot en forgyldt “grep”
- Mange fokuserer på sikkerhed (“tainted” data, buffer overflows)
- Nogle kræver ekstra servere blot til at analysere kildekoden
- Nogle er begrænset til få eller en platform

Andre C/C++ statisk analyse værktøjer

- Det største / mest velkendte spillere på markedet er:
  - Coverity
  - Polyspace
  - Fortify
  - Klocwork
  - Parasoft

Advarsler, udrag: 514

514 Unusual use of a Boolean -- An argument to an arithmetic operator (+ - / * %) or a bit-wise logical operator (| & ^) was a Boolean. This can often happen by accident as in:

```
if( flags & 4 == 0 )
```

where the ==, having higher precedence than &, is done first (to the puzzlement of the programmer).

Advarsler, udrag: 721

721 Suspicious use of ; -- A semi-colon was found immediately to the right of a right parenthesis in a construct of the form if(e);. As such it may be overlooked or confused with the use of semi-colons to terminate statements. The message will be inhibited if the ';' is separated by at least one blank from the ')'. Better, place it on a separate line. See also message 548.

Advarsler, udrag: 1725

1725 class member 'Symbol' is a reference -- There are a number of subtle difficulties with reference data members. If a class containing a reference is assigned, the default assignment operator will presumably copy the raw underlying pointer. This violates the principle that a reference's underlying pointer, once established, is never modified. Some compilers protect against this eventuality by refusing to create a default assignment operator for classes containing references. Similar remarks can be made about copy constructors. If you are careful about how you design your copy constructors and assignment operators, then references within classes can be a useful programming technique. They should not, however, be employed casually. [21, §2.1.3]

FlexeLint er aldrig glad (1)

```
#include <stdio.h>
int main() {
    printf("Hello world!\n");
    return 0;
}
```

flexelint -w4 on hello.cc

FlexeLint er aldrig glad (2)

```
int main() {
hello.cc  2  Note 970: Use of modifier or type 'int' outside of a typedef
hello.cc  2  Note 1917: empty prototype for definition, assumed '(void)'

                        _
    printf("Hello world!\n");
hello.cc  3  Note 1960: Violates MISRA C++ 2008 Required Rule 5-2-12, Array
type passed to function expecting a pointer
hello.cc  3  Warning 534: Ignoring return value of function 'printf(const
char*, ...)' (compare with line 341, file /usr/include/stdio.h)

    --- Wrap-up for Module: hello.cc

Note 966: Indirectly included header file '/usr/include/features.h' not used by
module 'hello.cc'
Note 966: Indirectly included header file '/usr/include/sys/cdefs.h' not used
by module 'hello.cc'
Note 966: Indirectly included header file '/usr/include/bits/wordsize.h' not
used by module 'hello.cc'
Note 966: Indirectly included header file '/usr/include/gnu/stubs.h' not used
by module 'hello.cc'
Note 966: Indirectly included header file '/usr/include/gnu/stubs-64.h' not
used by module 'hello.cc'
...
---snip---
```

Undertryk advarsel på en enkelt linje

```
static unsigned long hash_ip6_address(const in6_addr &ip_addr) {
    const uint32_t *p = (const uint32_t*)&ip_addr; //lint !e740
    return (unsigned long)p[0]+p[1]+p[2]+p[3];
}
```

Undertryk advarsel i en blok

```
void KeyTable::forget(Key *key) {
    //lint --e{818}   key could be const
    if(key->prev) {
        key->prev->next = key->next;
    } else {
        unsigned long hash_value = hashKey(key->key);
        unsigned long hash_index = hash_value%table_size;
        assert(table[hash_index]==key);
        table[hash_index] = key->next;
    }
    if(key->next) key->next->prev = key->prev;
}
```

Undertrykkelse af advarsler: assert()

- Velplacerede og korrekte assert()s kan hjælpe Flexelint
- FlexeLints value-tracking er ikke perfekt
  - modulus (%) forvirrer den ofte
  - Co-dependent variabler bliver ikke sporet
  - FlexeLint's kendskab til funktioner er ikke komplet

Før assert

```
#include <string.h>
size_t q2_len(const char *s) {
    const char *p = strchr(s,'q');
    if(p)
        p=strchr(p+1,'q');
    if(!p)
        p=strchr(s,'\0');
    return p-s;
}
```

Genererer for return statement:

```
Warning 613: Possible use of null pointer 'p' in left
argument to operator 'ptr-ptr'
```

Efter assert

```
#include <string.h>
#include <assert.h>
size_t q2_len(const char *s) {
    const char *p = strchr(s,'q');
    if(p)
        p=strchr(p+1,'q');
    if(!p) {
        p=strchr(s,'\0');
        assert(p);
    }
    return p-s;
}
```

Undertrykkelse af advarsler: et råd fra mig

- Fiks det *faktiske* problem i stedet for at skjule det
- Undertryk i en separat fil i stedet for I kildekoden (hvis muligt)
- Indsæt kommentar med advarselsteksten og hvorfor flexelint tager fejl og du har ret

Forkert/rigtig fix

```
void get_time_limit(time_t *result);

void foo() {
    uint32_t time_limit;
    get_time_limit((time_t*)&time_limit); //Note 1924: C-style cast
}
```

Forkert fix:

```
 get_time_limit(reinterpret_cast<time_t*>(&time_limit));
```

Rigtigt fix:

```
  time_t time_limit
```

eller:

```
  void get_time_limit(uint32_t *result);
```

Forkert/rigtig fix (2)

```
void log_string(char *str) {
    printf("%s\n",str);
}
```

På et tidspunkt begyndte compilere at brokke sig over konverteringen af “...“ strenge til non-const char*. Gammelt, forkert fix:

```
void foo() {
    log_string((char*)"Hello world"); //Info 1773: Attempt to
                                      //cast away const (or
                                      //volatile)
}
```

Rigtigt fix:

```
void log_string(const char *str) { ...
```

Tilpasning af Flexelint

Flexelint skal tilpasses til ens miljø:

- OS headerfiler
- Compiler-specialiteter
- Evt. Library headerfiler
- Undervejs finder man tit underlige ting. F.eks. dette lille “guldkorn” fra Linux' sys/types.h:

```
typedef unsigned int u_int8_t __attribute__ ((__mode__ (__QI__)));
```