Evaluación de código C

Ignacio Nieto Vidaurrázaga *inietv00@estudiantes.unileon.es*

Diseño y Programación seguras - Máster en Ciberseguridad

1. BLOQUE DLC: Reglas

Repaso de la REGLA 2: Declarations and Initialization (DCL) en el segun el SEI CERT C Coding Standard.

- DCL30-C. Declare objects with appropriate storage durations
- DCL31-C. Declare identifiers before using them
- DCL36-C. Do not declare an identifier with conflicting linkage classifications
- DCL37-C. Do not declare or define a reserved identifier
- DCL38-C. Use the correct syntax when declaring a flexible array member
- DCL39-C. Avoid information leakage when passing a structure across a trust boundary
- DCL40-C. Do not create incompatible declarations of the same function or object
- DCL41-C. Do not declare variables inside a switch statement before the first case label

1.1. Ejemplo 1

Revisa y evalua la siguiente traza de código.

1. Define la regla que se incumple y proón una alternativa más adecuada según el SEI CERT C.

```c
#include <stdio.h>
#include <stddef.h>

const char *p;

char *funcion1(void) {
  char array[10] ="Mi Cadena";
  /* Initialize array */
  return array;
}

void funcion2(void) {
  const char c_str[] = "Todo va bien";
  p = c_str;
}

void funcion3(void) {
  printf("%s\n", p);
}

int main(void) {

  p = funcion1();
  printf("%s\n",p)
  funcion2();
  funcion3();
  printf("%s\n", p);

  return 0;
}
```

La regla SEI CERT incumplida: **STR32-C**. Do not pass a non-null-terminated character sequence to a library function that expects a string. No se termina la cadena bien con \0.

La solución sería dar valor a las cadenas siempre con el carácter finalizador.

Otros apuntes:

- No se inicializa el puntero *p, sería recomendable inicializarlo por ejemplo a NULL.
- La función *funcion1 devuelve una variable local, cuando se le asigna a p en el main puede tomar cualquier valor.

Además al ejecutar RATS sobre el codigo obtengo una advertencia que nos viene a decir lo mismo:
/* Severity: High

Issue: printf
Check to be sure that the non-constant format string passed as argument 1 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. */

- Otro fallo menor es que falta “;” en línea 24.

1.2. Ejemplo 2

1. ¿Qué hace el siguiente segmento de código?
2. De haber algún problema ¿Podrías decir la línea en la que se encuentra?
3. Define la regla que se incumple y proón una alternativa correcta siguiendo el SEI CERT C.

.

```
1  #include <stdlib.h>
2
3  struct flexArrayStruct {
4    int num;
5    int data[1];
6  };
7
8  void func(size_t array_size) {
9    /* Space is allocated for the struct */
10   struct flexArrayStruct *structP
11     = (struct flexArrayStruct *)
12       malloc(sizeof(struct flexArrayStruct)
13            + sizeof(int) * (array_size - 1));
14   if (structP == NULL) {
15     /* Handle malloc failure */
16   }
17
18   structP->num = array_size;
19
20   /*
21    * Access data[] as if it had been allocated
22    * as data[array_size].
23    */
24   for (size_t i = 0; i < array_size; ++i) {
25     structP->data[i] = 1;
26   }
27 }
```

En este otro fragmento de código se incumple la regla **MEM33-C**. Use the correct syntax for flexible array members.

El problema se encontraría en las líneas 5 y 13 y la solución que propone la Carnegie Mellon University es declarar en la línea 5 → *int data[]* .

Y en la línea 13 se quitaría el 1 --> *malloc(sizeof(struct flexArrayStruct) + sizeof(int) * array_size);*

Otro punto importante a considerar sería la liberación del espacio reservado con *malloc*, mediante *free*.

1.3. Ejemplo 2

1. ¿Qué hace el siguiente segmento de código si invocamos la función **func** con un **0**?
2. De haber algún problema ¿Podrías decir la línea en la que se encuentra?
3. Crea un fichero con un main y ejecuta el segmento de código.
4. Propón una solución al ejemplo que cumpla con las normal del CMU
5. Realiza un análisis estático del código erróneo y copia en tu solución el resultado. Utiliza las herramientas:
   (a) rats
   (b) cppchecker
   (c) splint
   (d) vera++
   (e) valgrind

```c
#include <stdio.h>

extern void f(int i);

void func(int expr) {
  switch (expr) {
    int i = 4;
    f(i);
  case 0:
    i = 17;
  default:
    printf("%d\n", i);
  }
}
```

1. Sería el único caso en el que la salida esta definida y es 17.

2. El problema se encuentra en la línea 7 y 8 ya que se declaran variables dentro del *switch* y antes de la etiqueta *case*.

3. Al ejecutar en un *main* la funcion vemos el comportamiento anómalo descrito con cualquier entero diferente de 0.

4. Al incumplirse la regla **DCL41-C**. Do not declare variables inside a switch statement before the first case label, el SEI CERT propone la solución de declarar *int i = 4* y *f(i)* antes del *switch.*

```c
#include <stdio.h>

extern void f(int i);

void func(int expr) {
    int i = 4;
        f(i);
    switch (expr) {

    case 0:
        i = 17;
    default:
        printf("%d\n", i);
    }
}
```

5. Ya al compilar con gcc nos salta advertencia

```
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ gcc -g codigo_erroneo.c -o codigo_erroneo
codigo_erroneo.c: In function ‘func’:
codigo_erroneo.c:6:7: warning: statement will never be executed [-Wswitch-unreachable]
    6 |    int i = 4;                  /* Problema: declara variables y contiene senter
      |        ^
```

Ahora vamos revisando con las diferentes herramientas:

```
/* RATS */
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ rats codigo_erroneo.c
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing codigo_erroneo.c
Total lines analyzed: 20
Total time 0.000068 seconds
294117 lines per second
```

```
/* CPPCHECK */
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ cppcheck --std=c11 -v codigo_erroneo.c
Checking codigo_erroneo.c ...
Defines:
Undefines:
Includes:
Platform:Native
```

```
/* SPLINT */
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ splint codigo_erroneo.c +bounds -paramuse -varuse                    127 □

Splint 3.1.2 --- 21 Feb 2021

codigo_erroneo.c: (in function func)
codigo_erroneo.c:8:7: Fall through case (no preceding break)
  Execution falls through from the previous case (use /*@fallthrough@*/ to mark
  fallthrough cases). (Use -casebreak to inhibit warning)
codigo_erroneo.c:10:10: Fall through case (no preceding break)
codigo_erroneo.c:6:13: Statement after switch is not a case: int i = 4
  The first statement after a switch is not a case. (Use -firstcase to inhibit
  warning)
codigo_erroneo.c: (in function main)
codigo_erroneo.c:19:2: Path with no return in function declared to return int
  There is a path through a function declared to return a value on which there
  is no return statement. This means the execution may fall through without
  returning a meaningful result to the caller. (Use -noret to inhibit warning)
codigo_erroneo.c:4:6: Function exported but not used outside codigo_erroneo:
                          func
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   codigo_erroneo.c:13:1: Definition of func

Finished checking --- 5 code warnings
```

```
/* VERA++ */
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ vera++ codigo_erroneo.c
codigo_erroneo.c:1: no copyright notice found
codigo_erroneo.c:5: horizontal tab used
codigo_erroneo.c:6: horizontal tab used
codigo_erroneo.c:6: line is longer than 100 characters
codigo_erroneo.c:7: horizontal tab used
codigo_erroneo.c:8: trailing whitespace
codigo_erroneo.c:8: horizontal tab used
codigo_erroneo.c:9: horizontal tab used
codigo_erroneo.c:10: horizontal tab used
codigo_erroneo.c:11: horizontal tab used
codigo_erroneo.c:11: line is longer than 100 characters
codigo_erroneo.c:12: horizontal tab used
codigo_erroneo.c:12: closing curly bracket not in the same line or column
codigo_erroneo.c:13: closing curly bracket not in the same line or column
codigo_erroneo.c:19: closing curly bracket not in the same line or column
```

```
/* VALGRIND */
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ valgrind ./codigo_erroneo
==4073== Memcheck, a memory error detector
==4073== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4073== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==4073== Command: ./codigo_erroneo
==4073==
17
==4073== Conditional jump or move depends on uninitialised value(s)
==4073==    at 0x48D25C3: __vfprintf_internal (vfprintf-internal.c:1646)
==4073==    by 0x48BDE6A: printf (printf.c:33)
==4073==    by 0x109162: func (codigo_erroneo.c:11)
==4073==    by 0x10917D: main (codigo_erroneo.c:18)
==4073==
==4073== Use of uninitialised value of size 8
==4073==    at 0x48B801B: _itoa_word (_itoa.c:179)
==4073==    by 0x48D160C: __vfprintf_internal (vfprintf-internal.c:1646)
==4073==    by 0x48BDE6A: printf (printf.c:33)
==4073==    by 0x109162: func (codigo_erroneo.c:11)
==4073==    by 0x10917D: main (codigo_erroneo.c:18)
==4073==
==4073== Conditional jump or move depends on uninitialised value(s)
==4073==    at 0x48B802C: _itoa_word (_itoa.c:179)
==4073==    by 0x48D160C: __vfprintf_internal (vfprintf-internal.c:1646)
==4073==    by 0x48BDE6A: printf (printf.c:33)
==4073==    by 0x109162: func (codigo_erroneo.c:11)
```

```
==4073==    by 0x10917D: main (codigo_erroneo.c:18)
==4073==
==4073== Conditional jump or move depends on uninitialised value(s)
==4073==    at 0x48D2243: __vfprintf_internal (vfprintf-internal.c:1646)
==4073==    by 0x48BDE6A: printf (printf.c:33)
==4073==    by 0x109162: func (codigo_erroneo.c:11)
==4073==    by 0x10917D: main (codigo_erroneo.c:18)
==4073==
==4073== Conditional jump or move depends on uninitialised value(s)
==4073==    at 0x48D172C: __vfprintf_internal (vfprintf-internal.c:1646)
==4073==    by 0x48BDE6A: printf (printf.c:33)
==4073==    by 0x109162: func (codigo_erroneo.c:11)
==4073==    by 0x10917D: main (codigo_erroneo.c:18)
==4073==
17
==4073==
==4073== HEAP SUMMARY:
==4073==     in use at exit: 0 bytes in 0 blocks
==4073==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==4073==
==4073== All heap blocks were freed -- no leaks are possible
==4073==
==4073== Use --track-origins=yes to see where uninitialised values come from
==4073== For lists of detected and suppressed errors, rerun with: -s
==4073== ERROR SUMMARY: 7 errors from 5 contexts (suppressed: 0 from 0)
```

Como observamos en varias de las herramientas hemos recibido adevertencias relacionadas con el *switch* y la declaración de las variables.

2. BLOQUE DLC: Recomendaciones

Repaso de las Recomendaciones 2: Declarations and Initialization (DCL) en el segun el SEI CERT C Coding Standard

- DCL00-C. Const-qualify immutable objects
- DCL01-C. Do not reuse variable names in subscopes
- DCL02-C. Use visually distinct identifiers
- DCL03-C. Use a static assertion to test the value of a constant expression
- DCL04-C. Do not declare more than one variable per declaration
- DCL05-C. Use typedefs of non-pointer types only
- DCL06-C. Use meaningful symbolic constants to represent literal values
- DCL07-C. Include the appropriate type information in function declarators
- DCL08-C. Properly encode relationships in constant definitions
- DCL09-C. Declare functions that return errno with a return type of errno_t
- DCL10-C. Maintain the contract between the writer and caller of variadic functions
- DCL11-C. Understand the type issues associated with variadic functions
- DCL12-C. Implement abstract data types using opaque types
- DCL13-C. Declare function parameters that are pointers to values not changed by the function as const
- DCL15-C. Declare file-scope objects or functions that do not need external linkage as static
- DCL16-C. Use "L,"not "l,"to indicate a long value
- DCL17-C. Beware of miscompiled volatile-qualified variables
- DCL18-C. Do not begin integer constants with 0 when specifying a decimal value
- DCL19-C. Minimize the scope of variables and functions
- DCL20-C. Explicitly specify void when a function accepts no arguments
- DCL21-C. Understand the storage of compound literals
- DCL22-C. Use volatile for data that cannot be cached
- DCL23-C. Guarantee that mutually visible identifiers are unique

2.1. Ejercicio 1

- ¿Qué hace el siguiente segmento de código?
- ¿Para qué se utiliza la variable va_eol?
- Incorpora el segmento de código en un programa .c de tal forma que no encontremos nigún *warning* cuando compilamos en gcc con los siguientes parámetros (std=c11). Dado que es C, elimina aquellos que no aplican. Escribe en la respuesta aquellos que se ven afectados y son eliminados.

```c
enum { va_eol = -1 };

unsigned int average(int first, ...) {
  unsigned int count = 0;
  unsigned int sum = 0;
  int i = first;
  va_list args;

  va_start(args, first);

  while (i != va_eol) {
    sum += i;
    count++;
    i = va_arg(args, int);
  }

  va_end(args);
  return(count ? (sum / count) : 0);
}
```

A partir de este fragmento de codigo se podría incumplir la recomendación **DCL10-C:** Maintain the contract between the writer and caller of variadic functions.

1. La función *average* lo que hace es calcular la media de los valores positivos pasados por argumento con *sum/count.*

2. Se utiliza para controlar, la función procesará argumentos hasta que recibe como argumento el mismo valor que *va_eol* (es decir -1).

3. No se usan *-Wno-variadic-macros -Wno-parentheses -fdiagnostics-show-opt.*

2.2. Ejercicio 2

- ¿Qué hace el siguiente segmento de código?
- Comenta qué reglas/recomendaciones se están rompiendo aquí. Tambien entran reglas pasadas.
- Instala la herramienta **perf** para realizar el profiling de la aplicación. Se puede instalar con apt.
- El programa permite mostrar el código desensamblado de la aplicación, adjunta alguna captura.
- ¿Podrías decir cual es la instrucción que más tiempo de CPU requiere? Adjunta uns captura y describe la razón.

```c
#include <stdio.h>

unsigned long long int factorial(unsigned int i) {

    if(i <= 1) {
        return 1;
    }
    return i * factorial(i - 1);
}

int  main(int argc, char *argv[]) {
     int i = 12, j=3, f=0;

     if (argc==1){
          printf("Factorial of %d is %lld\n", i, factorial(i)
             );
     }
     else{
          j=atoi(argv[1]);
          for (f=0;f<j;f++){
                  printf("Factorial of %d is %lld\n", f,
                     factorial(f));
          }
     }

    return 0;
}
```

1. Calcula el factorial de numero recibido por parámetro, el número *12* → salida: *Factorial of 12 is 479001600.*

2. Se incumple la regla **MSC24-C:** Do not use deprecated or obsolescent functions. Seria conveniente usar *strtol()* en lugar de *atoi()*.

Se incumple también la recomendacion SEI CERT **DCL04-C:** Do not declare more than one variable per declaration. Hubiera sido mejor declarar e inicializar cada variable por separado → *int i = 12; int j = 3; int f = 0;*

Otros apuntes:

- Al compilar se recibe advertencia

```
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ gcc -g factorial.c -o factorial
factorial.c: In function ‘main’:
factorial.c:19:5: warning: implicit declaration of function ‘atoi’ [-Wimplicit-function-declaration]
  19 |   j=atoi(argv[1]);
     |   ^~
```

Y habria que incluir en el programa la siguiente librería → *#include <stdlib.h>*

3.

```
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ sudo perf report
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 9  of event 'cycles'
# Event count (approx.): 1693574
#
# Children      Self  Command    Shared Object      Symbol
# ........  ........  .........  .................  ................................
#
    55.71%    55.71%  factorial  [kernel.kallsyms]  [k] alloc_pages_vma
            |
            ---0x7f336fd4d0c5
               0x7f336fb6fb49
               asm_exc_page_fault
               exc_page_fault
               do_user_addr_fault
               handle_mm_fault
               alloc_pages_vma

    55.71%     0.00%  factorial  [unknown]          [k] 0x00007f336fd4d0c5
            |
```

- ¿Podrías decir cual es la instrucción que más tiempo de CPU requiere la siguiente traza de código? Adjunta uns captura y describe la razón.

```c
// fib.c
#include <stdio.h>
#include <stdlib.h>

int fib(int x) {
  if (x == 0) return 0;
  else if (x == 1) return 1;
  return fib(x - 1) + fib(x - 2);
}

int main(int argc, char *argv[]) {

  for (size_t i = 0; i < 45; ++i) {
    printf("%d\n", fib(i));
  }
  return 0;
}
```

Con el siguiente código se calcula la sucesión de Fibonacci hasta el elemento n.º 45:

```
┌──(mck6194@kali)-[~/master/DPS]
└─$ vi fib.c

┌──(mck6194@kali)-[~/master/DPS]
└─$ gcc -g fib.c -o fib

┌──(mck6194@kali)-[~/master/DPS]
└─$ time ./fib
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
10946
17711
28657
46368
75025
121393
196418
317811
514229
832040
1346269
2178309
3524578
5702887
9227465
14930352
24157817
39088169
63245986
102334155
165580141
267914296
433494437
701408733

real    18,95s
user    18,94s
sys 0,01s
cpu 99%
```

Ahora hacemos el informe con perf y lo visualizamos:

```
┌──(mck6194@kali)-[~/master/DPS]
└─$ sudo perf record ./fib
...
...
...
701408733
[ perf record: Woken up 11 times to write data ]
[ perf record: Captured and wrote 2,887 MB perf.data (75119 samples) ]
```

```
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ perf report
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 75K of event 'cycles'
# Event count (approx.): 53408119097
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  ....................
#
    15.88%  fib      fib                [.] 0x0000000000001136
    14.43%  fib      fib                [.] 0x0000000000001141
    13.55%  fib      fib                [.] 0x000000000000117d
     7.95%  fib      fib                [.] 0x000000000000113e
     6.77%  fib      fib                [.] 0x0000000000001145
     6.68%  fib      fib                [.] 0x0000000000001163
     5.63%  fib      fib                [.] 0x000000000000113a
     5.62%  fib      fib                [.] 0x0000000000001135
     4.90%  fib      fib                [.] 0x0000000000001177
     3.70%  fib      fib                [.] 0x0000000000001168
     3.53%  fib      fib                [.] 0x0000000000001152
     2.65%  fib      fib                [.] 0x000000000000116d
     1.76%  fib      fib                [.] 0x000000000000117e
     1.53%  fib      fib                [.] 0x0000000000001172
     1.43%  fib      fib                [.] 0x000000000000116a
     1.26%  fib      fib                [.] 0x0000000000001159
     1.11%  fib      fib                [.] 0x0000000000001179
     1.02%  fib      fib                [.] 0x000000000000114c
     0.18%  fib      fib                [.] 0x0000000000001139
     0.14%  fib      fib                [.] 0x000000000000114e
     0.03%  fib      fib                [.] 0x000000000000115b
     0.02%  fib      [kernel.kallsyms]  [k] 0xffffffffb18b1dba
     0.02%  fib      fib                [.] 0x0000000000001170
     0.01%  fib      fib                [.] 0x0000000000001154
```

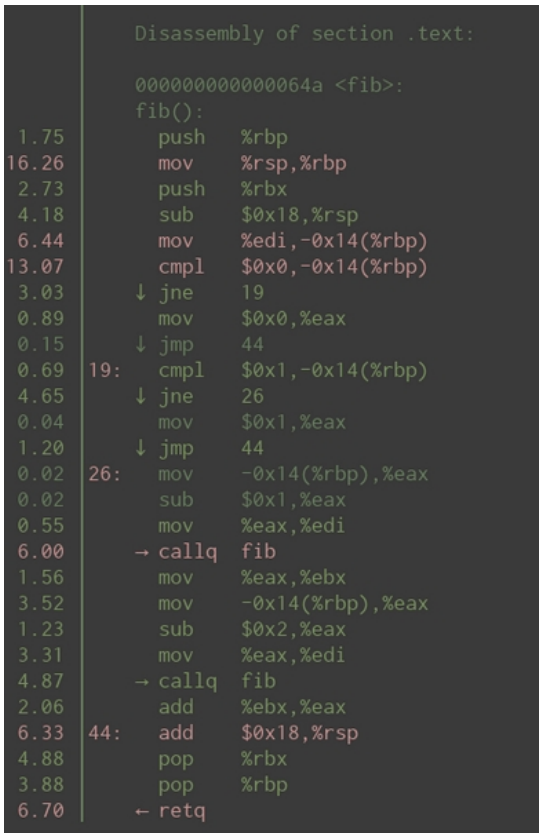
Realizando perf con el siguiente comando:

```
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ sudo perf record -g ./fib
...
...
701408733
[ perf record: Woken up 93 times to write data ]
[ perf record: Captured and wrote 23,182 MB perf.data (76460 samples) ]
```

Obtenemos el siguiente árbol:

```
┌──(mck6194㉿kali)-[~/master/DPS]
└─$ perf report
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 76K of event 'cycles'
# Event count (approx.): 53804115773
#
# Children      Self  Command  Shared Object      Symbol
# ........  ........  .......  .................  ....................
#
    100.00%     0.00%  fib      [unknown]          [.] 0x5541d68949564100
            |
            ---0x5541d68949564100
               0x7effea2a7e4a
               0x55d843ddb1a3
               |
               |--61.82%--0x55d843ddb168
               |          |
               |          |--38.23%--0x55d843ddb168
               |          |          |
               |          |          |--23.63%--0x55d843ddb168
               |          |          |          |
               |          |          |          |--14.62%--0x55d843ddb168
               |          |          |          |          |
               |          |          |          |          |--9.06%--0x55d843ddb168
               |          |          |          |          |          |
               |          |          |          |          |          |--5.62%--0x55d843ddb168
               |          |          |          |          |          |          |
               |          |          |          |          |          |          |--3.48%--0x55d843ddb168
               |          |          |          |          |          |          |          |
               |          |          |          |          |          |          |          |--2.16%--0x55d843ddb168
               |          |          |          |          |          |          |          |          |
               |          |          |          |          |          |          |          |          |--1.34%--0x55d843ddb168
               |          |          |          |          |          |          |          |          |          |
```

Por defecto perf solo obtiene información de tiempos. Probablemente queramos ver información tambie sobre llamadas o instrucciones y para ello hacemos el desensamblado.



Vemos que la mayoría de tiempo cosumido es en la instrucción *mov %rsp,%rbp* que son los registros responsables de stack (pila) y de frame pointers que contienen la direccion base de la función en arquitectura x86.

Como vemos se ha reflejado el conjunto de llamadas recursivas que se realiza en la funcion **fib** de Fibonacci y que tienen un gran coste para el sistema (si le pidiéramos a la función que calculara la sucesión para elementos más grandes probablemente produciría problemas de sistema y de consumo de CPU).

Este mismo problema pero a menor escala se producía con la funcion **factorial** al ser recursiva.