Secure Coding
II Parte

Le buone pratiche di Secure Coding secondo lo standard CERT

Il CERT Secure Coding Standards

- Sviluppato in modo collaborativo dal Cert CMU (il primo della storia)
- È possibile contribuire (formato e modalità) Wiki

Tre standard principali:

- **CERT Sun Microsystems Secure Coding Standard for Java**
- **CERT C Secure Coding Standard**
- **CERT C++ Secure Coding Standard**

Gli standard di secure coding

- Regole vs Raccomandazioni

Una buona pratica di programmazione è definite **regola** se

- La violazione della buona pratica risulterà in un security flaw che potrebbe generare una vulnerabilità sfruttabile.
- Esiste un insieme numerabile di condizioni eccezioni (o nessuna condizione) dove la violazione della regola è necessaria per assicurare il funzionamento corretto del programma.
- La conformità alla buona pratica può essere verificata.

Raccomandazione, linee guida suggerimenti

- L'applicazione migliora ragionevolmente la sicurezza
- Una o più delle condizioni sopra non è verificata

Livelli di priorità

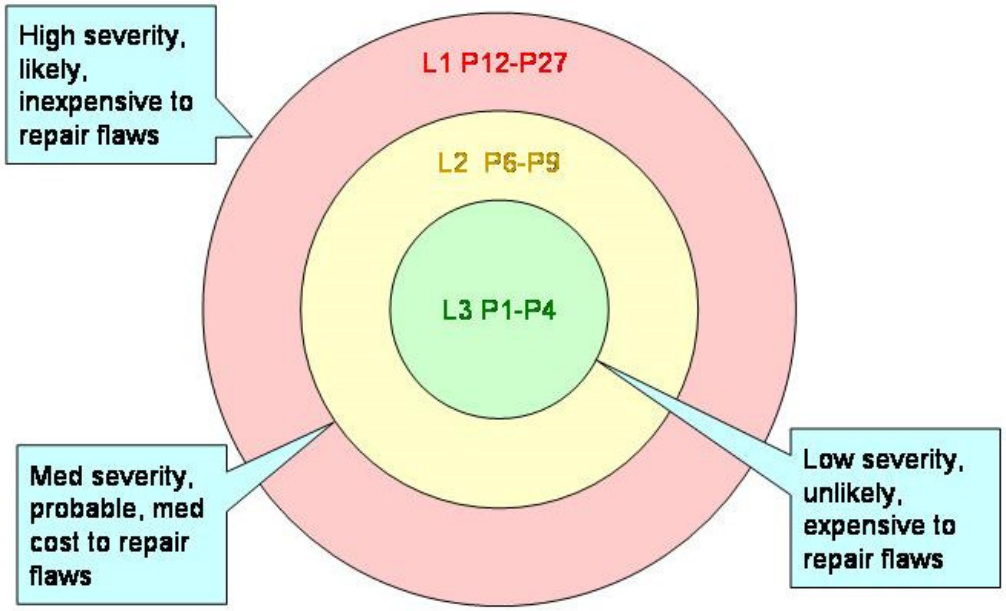
- Ad ogni regola o raccomandazione sono assegnati tre valori relativi a

- **severity** - how serious are the consequences of the rule being ignored;
  1 = low (denial-of-service attack, abnormal termination)
  2 = medium (data integrity violation, unintentional information disclosure)
  3 = high (run arbitrary code)
- **likelihood** - how likely is it that a flaw, introduced by ignoring the rule, could lead to an exploitable vulnerability;
  1 = unlikely
  2 = probable
  3 = likely
- **remediation cost** - how expensive is it to comply with the rule.
  1 = high (manual detection and correction)
  2 = medium (automatic detection / manual correction)
  3 = low (automatic detection and correction)

I valori sono poi moltiplicati loro per ottenere una misura di priorità

Livelli di compliance allo standard

- Le priorità di regole racc. determinano i livelli di aderenza allo standard da livello 1 (solo quelle ad alta priorità (12-27 punti) a livello 3(aderenza piena)



Metrica di Vulnerabilità del CERT

- Viene definita una metrica di vulnerabilità con un numero tra 0 e 180 tenendo conto di fattori quali:

- *Is information about the vulnerability widely available or known?*
- *Is the vulnerability being exploited in incidents reported to CERT or other incident response teams?*
- *Is the Internet Infrastructure at risk because of this vulnerability? (e.g., routers, name servers, critical Internet protocols)*
- *How many systems on the Internet are at risk from this vulnerability?*
- *What is the impact of exploiting the vulnerability?*
- *How easy is it to exploit the vulnerability?*
- *What are the preconditions required to exploit the vulnerability?*

La metrica non è una vera misura, ma più un ordinamento, nel senso che vulnerabilità 40 ad es. non è necessariamente il doppio di 20

Cert C Secure Coding Standard Categorie

- ***01. Preprocessor (PRE)***
- 02. Declarations and Initialization (DCL)
- ***03. Expressions (EXP)***
- 04. Integers (INT)
- 05. Floating Point (FLP)
- 06. Arrays (ARR)
- 07. Characters and Strings (STR)
- 08. Memory Management (MEM)
- 09. Input Output (FIO)
- 10. Environment (ENV)
- 11. Signals (SIG)
- 12. Error Handling (ERR)
- 13. Application Programming Interfaces (API)
- 49. Miscellaneous (MSC)
- 50. POSIX (POS)
- 99. The Void

PRE01-C. Use parentheses within macros around parameter names

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
PRE01-C	medium	probable	low	P12	L1

USARE PARENTESI NEI NOMI DI PARAMETRI
Problemi relativi al macro espansore, ambiguità precedenza operatori

Noncompliant Code Example

This `CUBE()` macro definition is noncompliant because it fails to parenthesize the parameter names.

```
#define CUBE(I) (I * I * I)
```

As a result, the invocation

```
int a = 81 / CUBE(2 + 1);
```

expands to

```
int a = 81 / (2 + 1 * 2 + 1 * 2 + 1);  /* evaluates to 11 */
```

which is clearly not the desired result.

Soluzione aderente allo standard

```
#define CUBE(I) ( (I) * (I) * (I) )
int a = 81 / CUBE(2 + 1);
```

La suite LDRA tool suite V 7.6.0 rileva questo tipo di violazione

PRE01-C. Use parentheses within macros around parameter names

USARE PARENTESI NEI NOMI DI PARAMETRI
Problemi relativi al macro espansore, ambiguità precedenza operatori

Noncompliant Code Example

This `CUBE()` macro definition is noncompliant because it fails to parenthesize the parameter names.

```
#define CUBE(I) (I * I * I)
```

As a result, the invocation

```
int a = 81 / CUBE(2 + 1);
```

expands to

```
int a = 81 / (2 + 1 * 2 + 1 * 2 + 1);  /* evaluates to 11 */
```

which is clearly not the desired result.

Soluzione aderente allo standard

```
#define CUBE(I) ( (I) * (I) * (I) )
int a = 81 / CUBE(2 + 1);
```

PRE07-C. Avoid using repeated question marks

- Problema: nello standard C99 tutti i seguenti trigrafi(seq.di 3 caratt.) sono sostituiti come un singolo carattere.

??=	#		??)	]		??!	|
??(	[		??'	^		??>	}
??/	\		??<	{		??-	~

Es. a++ non è eseguito poiché è

```
// what is the value of a now??/
a++;
```

nella stessa riga del commento(inserito \)

Soluzione compliant

```
// what is the value of a now? ?/
a++;
```

PRE07-C. Avoid using repeated question marks

??=	#		??)	]		??!	|
??(	[		??'	^		??>	}
??/	\		??<	{		??-	~

Non compliant:

```
size_t i = /* some initial value */;
if (i > 9000) {
   if (puts("Over 9000!??!") == EOF) {
     /* Handle Error */
   }
}
```

stampa *Over 9000!|*

se compilato C99

Soluzione aderente con concatenazione di stringhe

```
size_t i = /* some initial value */;
/* assignment of i */
if (i > 9000) {
   if (puts("Over 9000!?""?!") == EOF) {
     /* Handle Error */
   }
}
```

Stampa *Over 9000!??!* come voluto

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
PRE07-C	low	unlikely	medium	P2	L3

PRE11-C. Do not conclude a single statement macro definition with a semicolon

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
PRE11-C	medium	probable	low	P12	L1

- La frase viene stampata una sola volta a causa del ";" nel for anziché tre…

```
#define FOR_LOOP(n)  for(i=0; i<(n); i++);

int i;
FOR_LOOP(3)
{
  puts("Inside for loop\n");
}
```

The programmer expects to get the following output from the code:

```
Inside for loop
Inside for loop
Inside for loop
```

```
#define FOR_LOOP(n)  for(i=0; i<(n); i++)

int i;
FOR_LOOP(3)
{
  puts("Inside for loop\n");
}
```

DCL01-C. Do not reuse variable names in subscopes

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
DCL01-C	low	unlikely	medium	P2	L3

- Non inizializza la variabile globale *msg*

```
char msg[100];

void report_error(const char *error_msg) {
  char msg[80];
  /* ... */
  strncpy(msg, error_msg, sizeof(msg));
  return;
}

int main(void) {
  char error_msg[80];
  /* ... */
  report_error(error_msg);
  /* ... */
}
```

```
char system_msg[100];

void report_error(const char *error_msg) {
  char default_msg[80];
  /* ... */
  if (error_msg)
    strncpy(system_msg, error_msg, sizeof(system_msg));
  else
    strncpy(system_msg, default_msg, sizeof(system_msg));
  system_msg[ sizeof(system_msg) - 1] = '\0';
  return;
}

int main(void) {
  char error_msg[80];
  /* ... */
  report_error(error_msg);
  /* ... */
}
```

- Il programmatore

È forzato ad essere più preciso e descrittivo

- Rilevato da TLDRA tool suite Version 7.6.0, Splint Version 3.1.1,Compass/ROSE , Klocwork Version 8.0.4.16

DCL06-C. Use meaningful symbolic constants to represent literal values in program logic

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
DCL06-C	low	unlikely	medium	P2	L3

- Difficoltà per modificare le costanti numeri e stringa etc.
- Attenzione alla scelta della soluzione: *enum,macro o const*
- *Visibilità debugger, memoria, controllo tipi*
- Es. const unsigned int buffer_size = 256;

Method	Evaluated at	Consumes Memory	Viewable by Debuggers	Type Checking	Compile-time constant expression
Enumerations	compile time	no	yes	yes	yes
`const`-qualified	run time	yes	yes	yes	no
Macros	preprocessor	no	no	no	yes

DCL06-C. Use meaningful symbolic constants to represent literal values in program logic

- Il significato di 18 non è chiaro

```
/* ... */
if (age >= 18) {
   /* Take action */
}
else {
  /* Take a different action */
}
/* ... */
```

```
enum { ADULT_AGE=18 };
/* ... */
if (age >= ADULT_AGE) {
   /* Take action */
}
else {
  /* Take a different action */
}
/* ... */
```

- Il significato di 256 non è legato all'origine due es. compliant

```
char buffer[256];
/* ... */
fgets(buffer, 256, stdin);
```

```
enum { BUFFER_SIZE=256 };

char buffer[BUFFER_SIZE];
/* ... */
fgets(buffer, BUFFER_SIZE, stdin);
```

```
char buffer[256];
/* ... */
fgets(buffer, sizeof(buffer), stdin);
```

Espressioni

- EXP00-C. Use parentheses for precedence of operation
- EXP01-C. Do not take the size of a pointer to determine the size of the pointed-to type
- EXP02-C. Be aware of the short-circuit behavior of the logical AND and OR operators
- EXP03-C. Do not assume the size of a structure is the sum of the sizes of its members
- EXP04-C. Do not perform byte-by-byte comparisons involving a structure
- EXP05-C. Do not cast away a const qualification
- EXP06-C. Operands to the sizeof operator should not contain side effects
- EXP07-C. Do not diminish the benefits of constants by assuming their values in expressions
- EXP08-C. Ensure pointer arithmetic is used correctly
- EXP09-C. Use sizeof to determine the size of a type or variable
- EXP10-C. Do not depend on the order of evaluation of subexpressions or the order in which side effects take place
- EXP11-C. Do not apply operators expecting one type to data of an incompatible type
- EXP12-C. Do not ignore values returned by functions
- EXP13-C. Treat relational and equality operators as if they were nonassociative
- EXP14-C. Beware of integer promotion when performing bitwise operations on chars or shorts

EXP01-C. Do not take the size of a pointer to determine the size of the pointed-to type

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
EXP01-C	high	probable	medium	P12	L1

- Pericolo di buffer overflow

```
double *allocate_array(size_t num_elems) {
  double *d_array;

  if (num_elems > SIZE_MAX/sizeof(d_array)) {
    /* handle error condition */
  }
  d_array = (double *)malloc(sizeof(d_array) * num_elems);
  if (d_array == NULL) {
    /* handle error condition */
  }
  return d_array;
}
```

```
double *allocate_array(size_t num_elems) {
  double *d_array;

  if (num_elems > SIZE_MAX/sizeof(*d_array)) {
    /* handle error condition */
  }
  d_array = (double *)malloc(sizeof(*d_array) * num_elems);
  if (d_array == NULL) {
    /* handle error condition */
  }
  return d_array;
}
```

EXP30-C. Do not depend on order of evaluation between sequence points

- I sequence point sono i punti di valutazione di espressioni. Il C99 richiede che tra un sequence point e il successivo un oggetto sia modificato una sola volta

```
i = i + 1;
a[i] = i;
```

```
/* i is modified twice between sequence points */
i = ++i + 1;

/* i is read other than to determine the value to be stored */
a[i++] = i;
```

- Si consideri che *i* sia *0* nel seguente esempi

```
a = i + b[++i];
```

Due possibili interpretazioni e due sol.compl.

```
++i;
a = i + b[i];
```

```
a = i + b[i+1];
++i;
```

```
a = 0 + b[1];
```

```
a = 1 + b[1];
```

Altro es.

```
func(i++, i);
```

ha un comportamento indefinito due compliant

```
i++;
func(i, i);
```

```
j = i++;
func(j, i);
```

EXP30-C. Do not depend on order of evaluation between sequence points

- I sequence point sono i punti di valutazione di espressioni. Il C99 richiede che tra un sequence point e il successivo un oggetto sia modificato una sola volta

Possono rilevare questa **REGOLA**

- Splint Version 3.1.1
- GCC Compiler con il flag -**Wsequence-point**
- Compass/ROSE semplici violazioni

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
EXP30-C	medium	probable	medium	P8	L2

ARR31-C. Use consistent array notation across all source files

- Non mescolare dichiarazioni tipo **int *a** e **int a[]**

nei prototipi di funzione è identico, nei programmi *int a[]* è un tipo incompleto. Esempio: comportamento imprevisto in *a[0]='a';*

```
/* main.c source file */
#include <stdlib.h>

enum { ARRAYSIZE = 100 };

char *a;

void insert_a(void);

int main(void) {
  a = (char *)malloc(ARRAYSIZE);
  if (a == NULL) {
    /* Handle allocation error */
  }
  insert_a();
  return 0;
}
```

```
/* insert_a.c source file */
char a[];

void insert_a(void) {
  a[0] = 'a';
}
```

ARR31-C. Use consistent array notation across all source files

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
ARR31-C	high	probable	medium	P12	L1

- Soluzione complaint basata su tre file

```
/* insert_a.h include file */
enum { ARRAYSIZE = 100 };

extern char *a;
void insert_a(void);
```

```
/* insert_a.c source file */
#include "insert_a.h"
char *a;
void insert_a(void) {
   a[0] = 'a';
}
```

```
/* main.c source file */
#include <stdlib.h>
#include "insert_a.h"

int main(void) {
  a = (char *)malloc(ARRAYSIZE);
  if (a == NULL) {
    /* Handle allocation error */
  }
  insert_a();
  return 0;
}
```

- *Rilevato dalla suite* LDRA tool suite V 7.6.0
- **Regola anche per il C++**

Regole per gli array

- ARR30-C. Guarantee that array indices are within the valid range
- ARR31-C. Use consistent array notation across all source files
- ARR32-C. Ensure size arguments for variable length arrays are in a valid range
- ARR33-C. Guarantee that copies are made into storage of sufficient size
- ARR34-C. Ensure that array types in expressions are compatible
- ARR35-C. Do not allow loops to iterate beyond the end of an array
- ARR36-C. Do not subtract or compare two pointers that do not refer to the same array
- ARR37-C. Do not add or subtract an integer to a pointer to a non-array object
- ARR38-C. Do not add or subtract an integer to a pointer if the resulting value does not refer to a valid array element

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
ARR30-C	high	likely	high	P9	L2
ARR31-C	high	probable	medium	P12	L1
ARR32-C	high	probable	high	P6	L2
ARR33-C	high	likely	medium	P18	L1
ARR34-C	high	unlikely	medium	P6	L2
ARR35-C	high	likely	medium	P18	L1
ARR36-C	medium	probable	medium	P8	L2
ARR37-C	medium	probable	medium	P8	L2
ARR38-C	high	likely	medium	P18	L1

STR30-C. Do not attempt to modify string literals

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
STR30-C	low	likely	low	P9	L2

- Terminazione anormale e DoS

Es. comportamento imprevisto *p* è inizializzato con il puntatore alla costante stringa, mentre l'array *a* è allocato ed una copia della costante letterale vene copia nell'area di memoria dell'array

```
char a[] = "string literal";
a[0] = 'S';
```

```
char *p  = "string literal";
p[0] = 'S';
```

Esempio *mktemp* modifica la stringa passata come parametro

```
mktemp("/tmp/edXXXXXX");
```

```
static char fname[] = "/tmp/edXXXXXX";

mktemp(fname);
```

ENV03-C. Sanitize the environment when invoking external programs

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
ENV03-C	high	likely	high	P9	L2

- C99: i metodi per alterare l'ambiente sono *implementation defined*

Esempio. Posix NonComplaint

e Complaint, Notare l'uso di clearenv()

```
if (system("/bin/ls dir.`date +%Y%m%d`") == -1) {
  /* Handle error */
}
```

```
char *pathbuf;
size_t n;

if (clearenv() != 0) {
  /* Handle error */
}

n = confstr(_CS_PATH, NULL, 0);
if (n == 0) {
  /* Handle error */
}

if ((pathbuf = malloc(n)) == NULL) {
  /* Handle error */
}

if (confstr(_CS_PATH, pathbuf, n) == 0) {
  /* Handle error */
}

if (setenv("PATH", pathbuf, 1) == -1) {
  /* Handle error */
}

if (setenv("IFS", " \t\n", 1) == -1) {
  /* Handle error */
}

if (system("ls dir.`date +%Y%m%d`") == -1) {
  /* Handle error */
}
```

MSC02-C. Avoid errors of omission

- "then" eseguito solo se non è zero

```
if (a = b) {
 /* ... */
}
```

```
if (a == b) {
 /* ... */
}
```

```
if ((a = b) != 0) {
 /* ... */
}
```

- Una vera vulnerabilità del server Xwindow

```
/* First the options that are only allowed for root */
if (getuid() == 0 || geteuid != 0) {
  /* ... */
}
```

omissione di getuid() ritornava l'indirizzo della funzione che non è mai nullo quindi "or" sempre vero

Vari sistemi rilevano tra cui il GCC Compiler Version 4.4.0 con il flag **-Wall**

MSC20-C. Do not use a switch statement to transfer control into a complex block

```
int f(int i) {
  int j=0;
  switch (i) {
    case 1:
      for(j=0;j<10;j++) {
      // no break, process case 2 as well
    case 2: // switch jumps inside the for block
        j++;
      // no break, process case 3 as well
    case 3:
        j++;
      }
      break;
  default:
    // default action
    break;
  }
  return j;
}
```

```
int f(int i) {
  int j=0;
  switch (i) {
    case 1:
      // no break, process case 2 as well
    case 2:
      j++;
      // no break, process case 3 as well
    case 3:
      j++;
      break;
    default:
      // default action
      return j;
  }
  for(j++;j<10;j++) {
    j+=2;
  }
  return j;
}
```

MSC20-C. Do not use a switch statement to transfer control into a complex block

- Il Duff's Device ottimizzazione applicata ad una copia seriale di byte in un output per evitare di contare per ciascun byte

```
size_t count; /* must be nonzero */
char *to;     /* output destination */
char *from;   /* Points to count bytes to copy */

do {
  *to = *from++;        /* Note that the ''to'' pointer is NOT incremented */
} while (--count > 0);
```

```
int n = (count + 7) / 8;
switch (count % 8) {
case 0: do { *to = *from++;
case 7:      *to = *from++;
case 6:      *to = *from++;
case 5:      *to = *from++;
case 4:      *to = *from++;
case 3:      *to = *from++;
case 2:      *to = *from++;
case 1:      *to = *from++;
           } while (--n > 0);
}
```

- Fa count assegnamenti ma solo n confronti

MSC20-C. Do not use a switch statement to transfer control into a complex block

- Il Duff's Device ottimizzazione applicata ad una copia seriale di byte in un output per evitare di contare per ciascun byte

```
size_t count; /* must be nonzero */
char *to;     /* output destination */
char *from;   /* Points to count bytes to copy */

do {
  *to = *from++;        /* Note that the ''to'' pointer is NOT incremented */
} while (--count > 0);
```

```
int n = (count + 7) / 8;
switch (count % 8) {
case 0: do { *to = *from++;
case 7:      *to = *from++;
case 6:      *to = *from++;
case 5:      *to = *from++;
case 4:      *to = *from++;
case 3:      *to = *from++;
case 2:      *to = *from++;
case 1:      *to = *from++;
           } while (--n > 0);
}
```

```
int n = (count + 7) / 8;
switch (count % 8) {
case 0: *to = *from++; /* fall through */
case 7: *to = *from++; /* fall through */
case 6: *to = *from++; /* fall through */
case 5: *to = *from++; /* fall through */
case 4: *to = *from++; /* fall through */
case 3: *to = *from++; /* fall through */
case 2: *to = *from++; /* fall through */
case 1: *to = *from++; /* fall through */
}
while (--n > 0) {
  *to = *from++;
  *to = *from++;
  *to = *from++;
  *to = *from++;
  *to = *from++;
  *to = *from++;
  *to = *from++;
  *to = *from++;
}
```

- Fa count assegnamenti ma solo n confronti

Due esempi Java(1/2)

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
EXP03-J	low	unlikely	medium	P2	L3

- **EXP03-J. Do not compare string objects using equality or relational operators**
- Due riferimenti ad oggetti differenti con contenuti uguali

```
public class BadComparison {
  public static void main(String[] args) {
    String one = new String("one");
    String two = new String("one");
    if(one == two)
      System.out.println("Equal"); //not printed
  }
}
```

```
public class GoodComparison {
  public static void main(String[] args) {
    String one = new String("one");
    String two = new String("one");
    boolean result;
    if (one == null){
        result = two == null || two.equals(one);
    }
    else{
        result = one == two || one.equals(two);
    }
   System.out.println(result);
  }
}
```

- Usare *object1*.equals(*object2*)

Due esempi Java(2/2)

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
EXP02-J	medium	probable	medium	P8	L2

- **EXP02-J. Do not ignore values returned by methods.**
- Esempio. la stringa *original* non è aggiornata

```
public class Ignore {
  public static void main(String[] args) {
    String original = "insecure";
    original.replace( 'i', '9' );
    System.out.println (original);
  }
}
```

```
public class DoNotIgnore {
  public static void main(String[] args) {
    String original = "insecure";
    original = original.replace( 'i', '9' );
    System.out.println (original);
  }
}
```