Universidade do Minho
Escola de Engenharia

Ferramentas e técnicas de *Compiler Warnings*

GRUPO 1:

RICARDO PEREIRA

TIAGO RAMIRES

GRUPO 3:

ADRIANA MEIRELES

CARLA CRUZ

Introdução

- A grande maioria dos dispositivos com que lidamos necessita de proteção;
- É possível prevenir falhas fazendo **modelação de ameaças**;
- Existem outras formas de prevenir falhas de segurança no *software*?
- Medidas que se podem tomar durante a produção do código.
- ***Segmentation faults***, *compiler warnings* e **vulnerabilidades**.

Compiladores *GCC*

1. Funcionamento

- Conversão do código escrito numa linguagem em código máquina.
- "*gcc <nome do ficheiro> - o <nome do executável> - <opção1> - <opção2>*"
- Informa o utilizador da construção de **código impróprio** através de warnings.
- Estes warnings são constantemente ignorados pelos programadores.

2. GCC e G++

- O *GCC* está dividido em duas partes: ***backend*** e ***frontend***.
- *GCC* consegue compilar programas em *C++*, por exemplo.
- Existe um compilador especifico para compilar programas em *C++* - o *G++*.
- Determinadas opções de warnings só existem para algumas linguagens:
  - **-Wdeprecated-copy** - apenas para *C++*
  - **-Wsign-compare** - apenas para *C*

3. Diferenças entre ***errors*** e ***warnings***

- ***Errors*** impossibilitam a obtenção de um executável, impossibilitando também a execução.

- *Warnings* advertam para a construção de **código mau/irregular**:

  - relações entre variáveis de tipos diferentes;

  - utilização de funções perigosas;

  - declaração de variáveis que não são utilizadas;

- Existem opções que permitem **tornar *warnings* em *errors***.

4. Segmentation faults

- Um tipo de erros **bastante comum** e dos mais difíceis de solucionar.

- A que é que se devem os segmentation faults?

  - erro que ocorre **durante a execução** do programa quando este **acede a posições de memória que não deve**;

- Este tipo de erros **pode existir no programa mas nem sempre se manifestar**!

4. Segmentation faults

- Os *warnings* têm como função **prevenir este tipo de erros** (e não só).
- É frequente vermos um *warning* sempre que se utiliza a **função *gets***.
- Representam muitas vezes **vulnerabilidades de segurança no software**!

4. Segmentation faults

```c
#include <stdio.h>
extern int printfile(FILE *pointer);
extern int wc(FILE *pointer);

int main(int argc, char **argv)
{
        FILE *pointer;

        if( argc > 1 )
                pointer = fopen(argv[1], "r");
        else
                pointer = stdin;

        printfile(pointer);

        pointer = fopen(argv[1], "r");

        wc(pointer);

        fclose(pointer);
        return 0;
}
```

4. Segmentation faults

- **Heartbleed** – permitia a **obtenção de chaves privadas** armazenadas nas máquinas.

- ***GDB*** é uma ferramenta muito útil para análise de programas C.

```
*** stack smashing detected ***: <unknown> terminated

Program received signal SIGSEGV, Segmentation fault.
__GI_raise (sig=sig@entry=6) at ../sysdeps/unix/sysv/linux/raise.c:40
40        __libc_signal_block_app (&set);
(gdb) backtrace
#0  __GI_raise (sig=sig@entry=6) at ../sysdeps/unix/sysv/linux/raise.c:40
#1  0x00007ffff7a2b381 in __GI_abort () at abort.c:79
#2  0x00007ffff7a73a57 in __libc_message (action=action@entry=do_abort,
```

5. Vulnerabilidades

- Surgem com os problemas que vimos anteriormente:

  - **Stack buffer overflow** – executar funções que **não deviam ser executadas**;

  - **Heap buffer overflow** – leitura de **variáveis que o utilizador não pode ver;**

  - **Read overflow** – **leitura de dados da memória** (p.e., Heartbleed);

  - **Integer overflow** – o **valor guardado na memória é totalmente diferente** do pretendido.

6. Warnings

- **-w**
- **-Werror**
- **-Werror=<tipo de warning>**
- **-Wfatal-errors**
- **-Wpedantic e -pedantic**
- **-Wall**
- **-Wextra**

```
switch (cond)
  {
  case 1:
    a = 1;
    break;
  case 2:
    a = 2;
  case 3:
    a = 3;
    break;
  }
```

7. Warnings na prática - LOverflow2 (C++)

- Sem opções extra não apresenta warnings.

- Com as opções -Wall e –Wextra apresenta o seguinte warning:

```
LOverflow2.cpp: In function ‘int main()’:
LOverflow2.cpp:9:10: warning: variable ‘tests’ set but not used [-Wun
used-but-set-variable]
     int  tests[10];
          ^~~~~
```

7. Warnings na prática - RootExploit(C)

- Sem e com as opções, o warning relativo à **função gets** aparece sempre.

```
RootExploit.c: In function ‘main’:
RootExploit.c:10:5: warning: implicit declaration of function ‘gets’;
 did you mean ‘fgets’? [-Wimplicit-function-declaration]
     gets(buff);
     ^~~~
     fgets
/tmp/ccz2G5sV.o: In function `main':
RootExploit.c:(.text+0x37): warning: the `gets' function is dangerous
 and should not be used.
```

7. Warnings na prática - 0-simple (C)

- Sem e com as opções, o warning relativo à função gets aparece de novo neste programa.

```
0-simple.c: In function ‘main’:
0-simple.c:16:3: warning: implicit declaration of function ‘gets’; di
d you mean ‘fgets’? [-Wimplicit-function-declaration]
   gets(buffer);
   ^~~~
   fgets
/tmp/ccRkTyrP.o: In function `main':
0-simple.c:(.text+0x3e): warning: the `gets' function is dangerous an
d should not be used.
```

7. Warnings na prática - 2-functions (C)

- Um warning que apresenta uma irregularidade propositadamente forçada.

- **É necessário saber interpretá-los**!

```
2-functions.c:26:53: warning: format ‘%p’ expects argument of type ‘v
oid *’, but argument 2 has type ‘int (*)()’ [-Wformat=]
       printf("calling function pointer, jumping to %p\n", fp);
                                                    ~^
```

7. Warnings na prática - overflow (C)

- Um exemplo claro da **utilidade dos warnings**.
- O warning apresentado a seguir está **diretamente ligado a um segmentation fault**.
- Apenas é mostrado se a **opção -Wextra for ativada** ...

7. Warnings na prática - overflow (C)

```
overflow.c: In function ‘vulneravel’:
overflow.c:7:23: warning: comparison between signed and unsigned inte
ger expressions [-Wsign-compare]
         for (i = 0; i < x; i++) {
                       ^
overflow.c:8:31: warning: comparison between signed and unsigned inte
ger expressions [-Wsign-compare]
                 for (j = 0; j < y; j++) {
                               ^
overflow.c: In function ‘main’:
overflow.c:16:2: warning: ‘matriz’ is used uninitialized in this func
tion [-Wuninitialized]
  vulneravel(matriz, 5000000000, 5000000000, 0);
  ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

7. Frama-C

- Permite a verificação de programas de uma forma formal, matemática !

- Utiliza a lógica matemática fazer provas.

- Algumas utilidades:

  - declaração de invariantes de ciclo;

  - verifica cumprimento dos limites das variáveis;

  - permite estabelecer os inputs e outputs desejados no programa.

Conclusão

- Devem ser tidas em conta boas práticas para a construção de *software;*
- Estas ferramentas são essenciais para para garantir *softwares* fidedignos e funcionais;
- **A aplicação destas ferramentas não tem qualquer custo** !
- A redução de bugs é garantida;
- As boas práticas devem ser ensinadas desde logo **no período de formação** dos programadores.

Universidade do Minho
Escola de Engenharia

Tactical Threat Modeling

GRUPO 1:

RICARDO PEREIRA

TIAGO RAMIRES

GRUPO 3:

ADRIANA MEIRELES

CARLA CRUZ