Escola de Engenharia
Universidade do Minho

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado em Engenharia Informática
Engenharia de Segurança

Aula 9

20 de Abril de 2020

Grupo 1

Ricardo Pereira a73577

Tiago Ramires pg41101

Braga, 22 de Abril de 2020

1. Buffer Overflow

Pergunta P1.1 - *Buffer overflow* em várias linguagens

O programa *LOverflow2* consiste em criar um *array* de dez posições, pedir ao utilizador para introduzir uma determinada quantidade de números e por fim introduzir esses números. O *array* vai sendo preenchido desde a posição inicial até à última posição, sendo que quando o utilizador insere uma quantidade de números superior a 10, o programa devolve uma mensagem de erro indicando que o índice do *array* onde está a escrever não está dentro daquilo que foi definido.

Como podemos ver na figura 1.1, na execução do programa escrito em *java* ocorre uma exceção.

```
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte# javac LOverflow2.java
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte# java LOverflow2
Quantos números? 12
Introduza número: 0
Introduza número: 1
Introduza número: 2
Introduza número: 3
Introduza número: 4
Introduza número: 5
Introduza número: 6
Introduza número: 7
Introduza número: 8
Introduza número: 9
Introduza número: 10
Exception in thread "main" java.lang.A
rrayIndexOutOfBoundsException: 10
        at LOverflow2.main(LOverflow2.
java:18)
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte#
```

Figura 1.1: *Output* do programa *LOverflow2* em *java*.

Em *python*, o mesmo programa apresenta um erro, como é visível na figura 1.2.

```
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte# python LOverflow2.py
Quantos numeros? 12
Insira numero: 0
Insira numero: 1
Insira numero: 2
Insira numero: 3
Insira numero: 4
Insira numero: 5
Insira numero: 6
Insira numero: 7
Insira numero: 8
Insira numero: 9
Insira numero: 10
Traceback (most recent call last):
  File "LOverflow2.py", line 5, in <mo
dule>
    tests[i]=test
IndexError: list assignment index out
of range
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte#
```

Figura 1.2: *Output* do programa *LOverflow2* em *python*.

Por último, em *c++* ocorre uma exceção, como se mostra na figura 1.3.

```
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte# javac LOverflow2.java
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte# java LOverflow2
Quantos números? 12
Introduza número: 0
Introduza número: 1
Introduza número: 2
Introduza número: 3
Introduza número: 4
Introduza número: 5
Introduza número: 6
Introduza número: 7
Introduza número: 8
Introduza número: 9
Introduza número: 10
Exception in thread "main" java.lang.A
rrayIndexOutOfBoundsException: 10
        at LOverflow2.main(LOverflow2.
java:18)
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte#
```

Figura 1.3: *Output* do programa *LOverflow2* em *c++*.

As figuras anteriores demonstram ocorrências de *buffer overflows* do programa em questão nas diferentes linguagens.

Pergunta P1.2 - *Buffer overflow*

A vulnerabilidade que se verifica nos programas anteriores chama-se *stack based overflow*, uma vez que para concretizar um ataque explora-se a proximidade na *stack* dos endereços de memória das variáveis do programa. Para melhor compreendermos esta vulnerabilidade alteramos o código para que fossem imprimidos os endereços das variáveis.

Assim, no programa *RootExploit* existe um *buffer* com espaço para quatro carateres e um inteiro e sabemos também que a alocação destas duas variáveis é seguida, pelo que os endereços de memória de cada uma não podem ser muito distantes, como se verifica na figura 1.4. Posto isto, a função *gets* não verifica o tamanho do *buffer* para onde escreve, o que permite o utilizador escrever um número de carateres maior que o suportado, sendo esses carateres escritos, neste caso, no espaço de memória da variável *pass*. Ora, escrevendo-se um número de carateres igual ou superior a cinco, consegue-se assim obter a mensagem de confirmação.

```
RootExploit.c: In function 'main':
RootExploit.c:13:5: warning: implicit
declaration of function 'gets' [-Wimpl
icit-function-declaration]
     gets(buff);
     ^~~~
/tmp/ccJ1LALR.o: In function `main':
RootExploit.c:(.text+0x58): warning: t
he `gets' function is dangerous and sh
ould not be used.
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte# ./RootExploit
Address of pass: 0x7fffd7f8a85c
Address of buff: 0x7fffd7f8a858

 Insira a password de root:
aaaaa

 Password errada

 Foram-lhe atribuidas permissões de ro
ot/admin
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte#
```

Figura 1.4: *Output* do programa *RootExploit* em *c*.

No programa *0-simple*, a ideia é a mesma e como se pode ver na figura 1.5, os endereços de memória são ambos bastante próximos. Escrevendo em *buff* uma quantidade de carateres igual ou superior a setenta e sete, leva a que a variável *control* seja modificada, levando assim ao aparecimento da mensagem em questão.

```
a9/codigofonte# gcc -o 0-simple 0-simp
le.c
0-simple.c: In function ‘main’:
0-simple.c:19:3: warning: implicit dec
laration of function ‘gets’ [-Wimplici
t-function-declaration]
   gets(buffer);
   ^~~~
/tmp/ccnf9DoR.o: In function `main':
0-simple.c:(.text+0x5f): warning: the
`gets' function is dangerous and shoul
d not be used.
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte# ./0-simple
Address of control: 0x7fff92cb0c3c
Address of buffer: 0x7fff92cb0bf0
You win this game if you can change va
riable control'
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
a
YOU WIN!!!
root@CSI:~/EngSeg-master/TPraticas/Aul
a9/codigofonte#
```

Figura 1.5: *Output* do programa *0-simple* em *c*.

Pergunta P1.3 - *Read overflow*

Os *outputs* do programa presentes na figura 1.6 monstram a facilidade com que se obtém informação que está presente na memória. No caso em questão, a função *fgets* verifica, e bem, o tamanho do *buffer* para onde escreve e apesar de permitir a introdução de mais carateres que o que é suposto por parte do utilizador, escreve apenas aqueles que deve (quantidade igual à do tamanho do *buffer*). Contudo, o ciclo que está a imprimir o *buffer* está-se a basear apenas no número de carateres que o utilizador diz que vai introduzir, não verificando se ultrapassa ou não o tamanho desse *buffer*. Na figura a seguir é possível ver esse tipo de falhas que permitem o acesso a informação que não deveria ser visualizada pelo utilizador se este introduzir, por exemplo, 120 na quantidade de carateres a introduzir - são lidas cento e vinte posições e não cem, como era suposto.

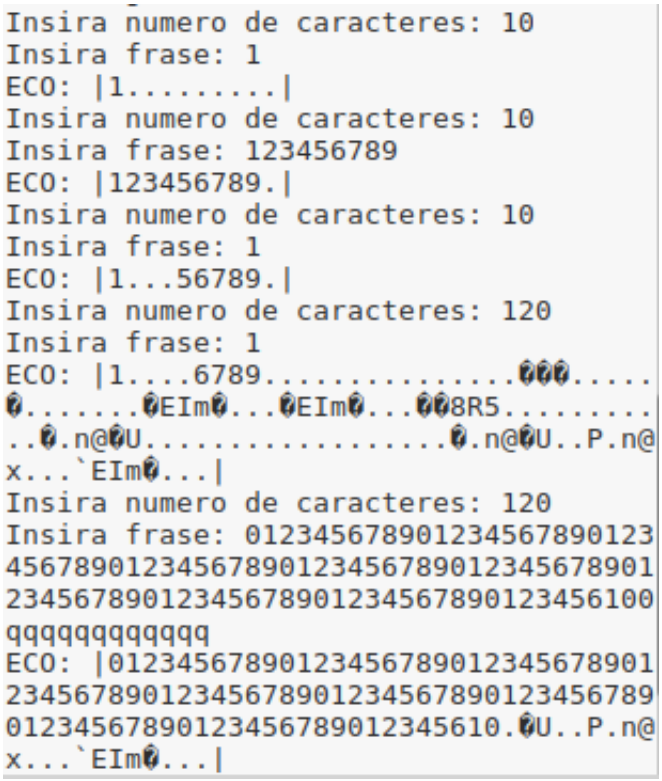

Figura 1.6: *Output* do programa *ReadOverflow* em *c*.

Experiência 1.4 - *Buffer overflow* em várias linguagens

O programa *LOverflow3* pede ao utilizador uma quantidade de números a guardar num *array* de inteiros (que tem espaço para dez valores) e preenche-o desde a primeira posição, onde insere essa quantidade. As posições seguintes equivalem, cada uma, à anterior menos uma unidade, pelo que a última posição preenchida do *array* terá o valor 1. De seguida é pedido ao utilizador para dizer qual a posição do *array* que gostava de visualizar, sendo-lhe devolvido o valor.

O programa em *java* foi testado em quatro aspetos:

- quantidade de valores menor que o tamanho do *array* e o índice da posição pedido menor que a quantidade de valores - o comportamento é o esperado;
- quantidade de valores menor que o tamanho do *array* e o índice da posição pedido maior que a quantidade de valores e menor que o tamanho do *array* - o programa inicializou automaticamente o *array* com o valor 0 em todas as posições e por esse motivo temos o resultado em questão, que não deveria ser mostrado ao utilizador;
- quantidade de valores menor que o tamanho do *array* e o índice da posição pedido maior que a quantidade de valores e que o tamanho do *array* - ocorre uma exceção por acesso a posições que não pertencem ao array;
- quantidade de valores maior que o tamanho do *array* e o índice da posição pedido maior que a quantidade de valores e que o tamanho do *array* - ocorre uma exceção por acesso a posições que não pertencem ao array;

```
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# java LOverflow3
Quantos valores quer guardar no array?
5
Que valor deseja recuperar?
0
O valor é 5
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# java LOverflow3
Quantos valores quer guardar no array?
5
Que valor deseja recuperar?
7
O valor é 0
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# java LOverflow3
Quantos valores quer guardar no array?
9
Que valor deseja recuperar?
15
Exception in thread "main" java.lang.Ar
rayIndexOutOfBoundsException: 15
        at LOverflow3.main(LOverflow3.j
ava:21)
```

Figura 1.7: *Output* do programa *LOverflow3* em *java*.

```
5
Que valor deseja recuperar?
7
O valor é 0
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# java LOverflow3
Quantos valores quer guardar no array?
9
Que valor deseja recuperar?
15
Exception in thread "main" java.lang.Ar
rayIndexOutOfBoundsException: 15
        at LOverflow3.main(LOverflow3.j
ava:21)
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# java LOverflow3
Quantos valores quer guardar no array?
15
Exception in thread "main" java.lang.Ar
rayIndexOutOfBoundsException: 10
        at LOverflow3.main(LOverflow3.j
ava:15)
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte#
```

Figura 1.8: *Output* do programa *LOverflow3* em *java*.

O programa em *python* foi testado nos mesmos quatro aspetos:

- quantidade de valores menor que o tamanho do *array* e o índice da posição pedido menor que a quantidade de valores - o comportamento é o esperado;
- quantidade de valores menor que o tamanho do *array* e o índice da posição pedido maior que a quantidade de valores e menor que o tamanho do *array* - o programa inicializou automaticamente o *array* com o valor *None* em todas as posições e por esse motivo temos o resultado em questão, que não deveria ser mostrado ao utilizador;
- quantidade de valores menor que o tamanho do *array* e o índice da posição pedido maior que a quantidade de valores e que o tamanho do *array* - ocorre um erro por acesso a posições que não pertencem ao *array*;
- quantidade de valores maior que o tamanho do *array* e o índice da posição pedido maior que a quantidade de valores e que o tamanho do *array* - ocorre um erro por acesso a posições que não pertencem ao *array*;

```
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# python LOverflow3.py
Quantos valores quer guardar no array?
5
Que valor deseja recuperar? 0
O valor e  5
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# python LOverflow3.py
Quantos valores quer guardar no array?
5
Que valor deseja recuperar? 7
O valor e  None
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# python LOverflow3.py
Quantos valores quer guardar no array?
9
Que valor deseja recuperar? 15
O valor e
Traceback (most recent call last):
  File "LOverflow3.py", line 6, in <mod
ule>
    print 'O valor e ', str(vals[which]
)
IndexError: list index out of range
```

Figura 1.9: *Output* do programa *LOverflow3* em *python*.

```
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# python LOverflow3.py
Quantos valores quer guardar no array?
9
Que valor deseja recuperar? 15
O valor e
Traceback (most recent call last):
  File "LOverflow3.py", line 6, in <mod
ule>
    print 'O valor e ', str(vals[which]
)
IndexError: list index out of range
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# python LOverflow3.py
Quantos valores quer guardar no array?
15
Traceback (most recent call last):
  File "LOverflow3.py", line 4, in <mod
ule>
    vals[i] = count-i
IndexError: list assignment index out o
f range
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte#
```

Figura 1.10: *Output* do programa *LOverflow3* em *python*.

O programa em *c++* foi submetido aos mesmos quatro testes:

- quantidade de valores menor que o tamanho do *array* e o índice da posição pedido menor que a quantidade de valores - o comportamento é o esperado;
- quantidade de valores menor que o tamanho do *array* e o índice da posição pedido maior que a quantidade de valores e menor que o tamanho do *array* - o programa inicializou automaticamente o *array* com o valor 0 em todas as posições e por esse motivo temos o resultado em questão, que não deveria ser mostrado ao utilizador;
- quantidade de valores menor que o tamanho do *array* e o índice da posição pedido maior que a quantidade de valores e que o tamanho do *array* - não ocorre qualquer erro ou exceção, apresentando o valor 0 que não deveria ser apresentado;
- quantidade de valores maior que o tamanho do *array* - o utilizador fica indefinidamente à espera do próximo passo, não havendo informação de qualquer erro ou exceção;

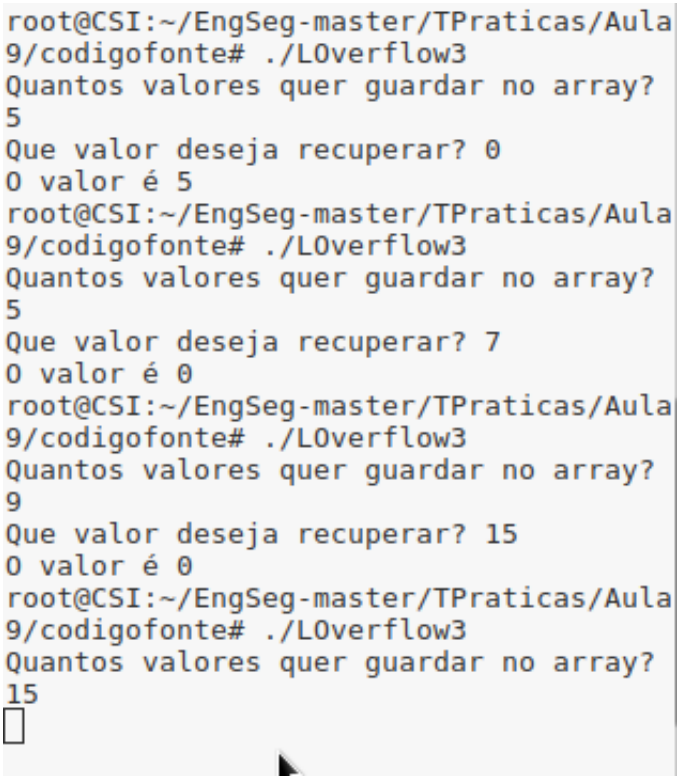

Figura 1.11: *Output* do programa *LOverflow3* em *c++*.

Este programa apresenta várias falhas e as mesmas seriam resolvidas muito facilmente com duas medidas de segurança:

- limitar o valor de *count* relativamente ao tamanho do *array*;
- limitar o valor de *which* relativamente ao valor de *count*.

Experiência 1.5 - *Buffer overflow* em várias linguagens

O programa *ReadTemps* lê valores de temperatura colocados num ficheiro. Para cada linguagem, primeiro testou-se o ficheiro com dez valores e depois com doze sendo que o *array* que armazena estes valores tem sempre disponíveis apenas dez posições.

A figura 1.12 mostra os *outputs* para situação descrita anteriormente

```
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# javac ReadTemps.java
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# java ReadTemps
Foram lidas 10 temperaturas.
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# java ReadTemps
Exception in thread "main" java.lang.Ar
rayIndexOutOfBoundsException: 10
        at ReadTemps.main(ReadTemps.jav
a:15)
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# python ReadTemps.py
Foram lidas 10 temperaturas.
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# python ReadTemps.py
Traceback (most recent call last):
  File "ReadTemps.py", line 6, in <modu
le>
    temps[numTemps] = float(i)
IndexError: list assignment index out o
f range
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte#
```

Figura 1.12: *Output* do programa *ReadTemps* em *java* e em *python* sem correção.

A figura 1.13 mostra os *outputs* para situação descrita com as devidas correções.

```
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# javac ReadTemps.java
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# java ReadTemps
Foram lidas 10 temperaturas.
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte# python ReadTemps.py
Foram lidas 10 temperaturas.
root@CSI:~/EngSeg-master/TPraticas/Aula
9/codigofonte#
```

Figura 1.13: *Output* do programa *ReadTemps* em *java* e em *python* com correção.

Pergunta P1.4

O programa *1-match* é bastante simples de compreender: recebe um argumento dado pelo utilizador (um *array* de carateres) e aloca duas variáveis - um inteiro *control* e um *array* de *chars buffer* - sendo que para esta última é copiada a informação introduzia pelo utilizador. No final do programa, se a variável *control* tiver o valor *0x61626364*, é impressa a mensagem *"Congratulations (...)"* senão é impressa a mensagem *"Try again (...)"*.

Para alterar esta variável utilizou-se o *gdb* para se encontrarem os endereços das variáveis *control* e *buffer*, sendo estes *0x7ffceddfb87c* e *0x7ffceddfb830*, respetivamente. Fazendo as contas, é necessário avançar setenta e seis endereços para nos posicionarmos naquele que está afeto à variável *control*, ou seja é necessário escrever setenta e seis carateres e depois escrever os carateres que correspondem a *0x61626364*, que no código *ASCII* fazem corresponder a *dcba*,

tendo em conta que a nossa máquina é *little endian*. Com estes passos escreve-se na variável o valor pretendido, como se vê na figura 1.14.

```
root@CSI:~/EngSeg-master/TPraticas/Aula9/codigofonte# gcc 1-match.c -g -o 1-match
1-match.c: In function ‘main’:
1-match.c:15:7: warning: implicit declaration of function ‘errx’ [-Wimplicit-funct
ion-declaration]
       errx(1, "please specify an argument\n");
       ^~~~
root@CSI:~/EngSeg-master/TPraticas/Aula9/codigofonte# ./1-match aaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaadcba
You win this game if you can change variable control to the value 0x61626364'
Congratulations, you win!!! You correctly got the variable to the right value
root@CSI:~/EngSeg-master/TPraticas/Aula9/codigofonte#
```

Figura 1.14: *Output* do programa *1-match* em *c*.

Pergunta P1.5 - *Buffer overflow* na *Heap*

Como foi visto na videoaula, o *input* introduzido pelo utilizador no programa, caso excedesse um determinado tamanho, conseguia modificar uma variável que deveria estar fora do controlo do utilizador. Esse tipo de problemas pode ser facilmente mitigado se se tiverem em atenção os limites das variáveis alocadas.

Assim, decidiu-se adotar as seguintes medidas:

- declarar uma variável *size* que controla o tamanho das alocações dinâmicas que são feitas - sempre que se pretender uma alocação daquele tamanho, usa-se a variável em vez do valor, premitindo assim o uso dessa variável para outras validações;
- controlo do número de argumentos introduzidos;
- controlo do comprimento da *string* submetida pelo utilizador que deve estar entre determinados valores (definidos pelo tamanho do *array* para onde vai ser copiada);
- verificação dos tamanhos das *strings* que são copiadas para excluir a hipótese de *overflow*.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    int size = 10;

    if (argc != 2 || strlen(argv[1]) >= size){
        printf("Os argumentos do programa são inconsistentes.
\n");
        return 0;
    }

    char *dummy = (char *) malloc (sizeof(char) * size);
    char *readonly = (char *) malloc (sizeof(char) * size);

    if(strlen("laranjas") < size)
        strcpy(readonly, "laranjas");
    if(strlen(argv[1]) < size)
        strcpy(dummy, argv[1]);
    printf("%s\n", readonly);
}
```

Figura 1.15: Correção do programa *overflowHeap.1*.

Pergunta P1.6 - *Buffer overflow* na *Stack*

Aparentemente, o programa em questão abre um ficheiro, lê o seu conteúdo, e copia-o para uma *string str* que posteriormente é passada a outra função. Por fim, essa função copia o conteúdo desta *string* para outra e o programa devolve uma mensagem de sucesso. Contudo, executando o programa obtemos um *segmentation fault*.

Após uma análise no *gdb*, descobriu-se que existem bastantes motivos para este *segmentation fault*, entre os quais:

- não verificação da existência do ficheiro que se tenta abrir - agora verifica-se o valor de retorno da função *fopen*, que caso retorne *NULL*, não executa as instruções associadas a esse ficheiro;
- uso do mesmo valor múltiplas vezes no ficheiro - guarda-se este valor numa variável e começa a usar-se a variável nos locais onde esse valor ocorria;
- não era feita a verificação do tamanho das *strings* envolvidas na operação *strcpy* - agora, caso a *string* destino seja menor que a *original*, a cópia não é feita.

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(char *str)
{
    int size = 24;
    char buffer[size];
    /* The following statement has a buffer overflow problem */
    if(strlen(str) < strlen(buffer))
        strcpy(buffer, str);
    return 1;
}

int main(int argc, char **argv)
{
    int size = 517;
    char str[size];
    FILE *badfile;
    if((badfile = fopen("badfile", "r")) != NULL){
        fread(str, sizeof(char), size, badfile);
        bof(str);
    }
    printf("Returned Properly\n");
    return 1;
}
```

Figura 1.16: Correção do programa *stack*.

Uma nova execução do programa retornou a mensagem *"Returned Properly"*, sendo agora o bom funcionamento do programa corroborado por outras ferramentas de verificação de boas práticas de programação disponibilizados pelo *gcc*.