Programowanie strukturalne - Wykład 5

Pamięć wirtualna

Pamięć wirtualna – mechanizm zarządzania pamięcią komputera zapewniający procesowi wrażenie pracy w jednym, dużym, ciągłym obszarze pamięci operacyjnej podczas, gdy fizycznie może być ona pofragmentowana, nieciągła i częściowo przechowywana na urządzeniach pamięci masowej. Systemy korzystające z tej techniki ułatwiają tworzenie rozbudowanych aplikacji oraz poprawiają wykorzystanie fizycznej pamięci RAM w systemach wielozadaniowych.

ASLR

ASLR (Address Space Layout Randomization) tłumaczony jest jako mechanizm losowego generowania lokalizacji alokacji pamięci wirtualnej.

Wykonywanie czynności przedstawionych na dalszych slajdach związanych z ASLR może oznaczać narażenie komputera na niebezpieczeństwo i podatność na ataki. Wykonanie tych działań nie jest zalecane i robione tylko na własną odpowiedzialność.

Exploit Protection

Funkcja Exploit Protection jest wbudowana w system Windows 10 w celu zabezpieczenia urządzenia przed atakami. Twoje urządzenie jest od razu skonfigurowane za pomocą ustawień ochrony, które są odpowiednie dla większości użytkowników.

Ustawienia funkcji Exploit Protection

Dowiedz się więcej

Wymuś losowe generowanie obrazów (obowiązkowa funkcja ASLR)
Wymuś relokację obrazów, które nie zostały skompilowane z użyciem przełącznika /DYNAMICBASE

Włączone domyślnie

Generuj losowo alokacje pamięci (funkcja ASLR „od dołu do góry")
Generuj losowo lokalizacje alokacji pamięci wirtualnej.

Włączone domyślnie

Funkcja ASLR o wysokiej entropii
Zwiększ zmienność podczas używania ustawienia Generuj losowo alokacje pamięci (funkcja ASLR „od dołu do góry").

Włączone domyślnie

Referencje w C++

Parę kolejnych informacji będzie dotyczyło języka C++.
Omawiane konstrukcje nie będą na egzaminie i nie można ich bezpośrednio przenieść do języka C.

Referencje są aliasami (innymi nazwami) dla zmiennych.
Referencja musi być zainicjalizowana podczas deklaracji i nie można jej zmienić, aby wskazywała na inną zmienną.
Referencje nie mogą mieć wartości “null”.

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 10;
    int &b = a;
    b = 20;
    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    return 0;
}
```

Wskaźniki są zmiennymi, które przechowują adresy innych zmiennych w pamięci. Wskaźniki mogą być zmieniane, aby wskazywały na różne zmienne lub przyjmowały wartość “null”. Wskaźniki można deklarować bez inicjalizacji, ale zaleca się inicjalizować je wartością “null” lub adresem zmiennej.

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 10;
    int *p = &a;
    *p = 20;
    cout << "a: " << a << endl;
    cout << "*p: " << *p << endl;
    int c = 30;
    p = &c;
    cout << "*p: " << *p << endl;
    return 0;
}
```

Referencje i wskaźniki a przekazywanie do funkcji

```cpp
#include <iostream>

using namespace std;

void funkcja_przyjmujaca_referencje(int &ref)
{
    ref = 50;
}

void funkcja_przyjmujaca_wskaznik(int *ptr)
{
    if (ptr) {
        *ptr = 100;
    }
}

int main()
{
    int a = 10;
    int b = 20;
    funkcja_przyjmujaca_referencje(a);
    cout << "a=" << a << endl;
    funkcja_przyjmujaca_wskaznik(&b);
    cout << "b=" << b << endl;
    return 0;
}
```

System szesnastkowy

https://pl.wikipedia.org/wiki/Szesnastkowy_system_liczbowy

Poprawnie:

```
1  int num=0;
2  int *pi = &num;
```

lub

```
1  int num = 0;
2  int*pi;
3  pi=&num;
```

Wątpliwe dla niektórych kompilatorów

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

int main()
{
    int num = 0;
    int*pi;
    pi= num;
    return 0;
}
```

Wskaźnik na stałą wartość, a stały wskaźnik

Wskaźnik na stałą wartość:

```
1   const int *a;
2   int const * a;
```

Stały wskaźnik:

```
1  int * const b;
```

Stały wskaźnik na stałą wartość:

```
1  int const * const c
```

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

int main()
{
    int i=0;
    const int *a=&i;
    int * const b=&i;
    int const * const c=&i;
    *a = 1;  /* kompilator zaprotestuje */
    *b = 2;  /* ok */
    *c = 3;   /* kompilator zaprotestuje */
    a = b;   /* ok */
    b = a;   /* kompilator zaprotestuje */
    c = a;   /* kompilator zaprotestuje */
    return 0;
}
```

Funkcja `malloc`

```
1  void *malloc(size_t size);
```

Funkcja służy do dynamicznego rezerwowania miejsca w pamięci. Gdy funkcja zostanie wywołana, w przypadku sukcesu zwróci wskaźnik do nowo zarezerwowanego miejsca w pamięci; w przypadku błędu zwraca wartość NULL.

Funkcja `free`

```
1  void free(void *ptr);
```

Funkcja free zwalnia blok pamięci wskazywany przez ptr wcześniej przydzielony przez `malloc`. Jeżeli ptr ma wartość NULL funkcja nie robi nic.

Rozmiar `int`

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

int main()
{
    printf("%Iu\n",sizeof(int)); //Windows
    //printf("%zu\n",sizeof(int)); //linux, os x
    printf("%Iu\n",sizeof(int*));
    printf("%Iu\n",sizeof(int**));
    return 0;
}
```

Wskaźniki na funkcję

```
1  typ_zwracany (*nazwa_wsk)(typ1 arg1, typ2 arg2);
```

```c
#include <stdio.h>

 int suma (int lhs, int rhs)
 {
   return lhs+rhs;
 }

 int main ()
 {
   int (*wsk_suma)(int a, int b);
   wsk_suma = suma;
   printf("4+5=%d\n", wsk_suma(4,5));
   return 0;
 }
```

Inaczej - bez nazwanych argumentów.

```c
#include <stdio.h>

 int suma (int lhs, int rhs)
 {
   return lhs+rhs;
 }

 int main ()
 {
   int (*wsk_suma)(int, int);
   wsk_suma = suma;
   printf("4+5=%d\n", wsk_suma(4,5));
   return 0;
 }
```

Jaka różnica?

```
1  int * wsk1();
2  int (*wsk2)();
3  int *(*wsk3)();
```

1. Funkcja zwracająca wskaźnik.
2. Wskaźnik na funkcję.
3. Wskaźnik na funkcję, zwracającą wskaźnik.

Wskaźnik na funkcję jako argument funkcji

```c
#include <stdio.h>

int dodaj(int a, int b)
{
    return a + b;
}

int odejmij(int a, int b)
{
    return a - b;
}

int wykonaj_operacje(int (*operacja)(int, int), int a, int b)
{
    return operacja(a, b);
}

int main()
{
```

Inne typy liczbowe?

Pełna analogia.

Duży błąd merytoryczny - 2 na egzaminie(!)

Dereferencja niezainicjalizowanych wskaźników:

```c
#include <stdio.h>
int main(void)
{
    int *wsk; // niezainicjalizowany wskaznik
    *wsk = 5;
    return 0;
}
```

Bibliografia

- Richard Reese, Wskaźniki w języku C, Wydawnictwo Helion 2014.
- https://pl.wikibooks.org/wiki/C/Wska%C5%BAniki, dostęp online 15.03.2020.
- http://wazniak.mimuw.edu.pl/index.php?title=Wst%C4%99p_do_programowania_w_j%C4%99zyku_C/Wska%C5%BAniki, dostęp online 15.03.2020.
- https://pl.wikibooks.org/wiki/C/Wska%C5%BAniki_-_wi%C4%99cej, dostęp online 15.03.2020.
- Stephen Prata, Język C. Szkoła programowania. Wydanie VI, Wyd. Helion, 2016.
- https://pl.wikibooks.org/wiki/C/Tablice, dostęp online 20.03.2020.
- https://pl.wikibooks.org/wiki/C/Tablice_-_wi%C4%99cej, dostęp online 20.03.2020.