Лекция 16

3 апреля

«Заглянуть за горизонт»

```c
void f(int i) {
   int a[3] = {1, 2, 3};
   printf("%x\n", a[i]);
}
```

i	Вывод на экран	
0	1	массив a
1	2	
2	3	
3	b7ff1030	«мусор»
4	8049ff4	
5	bffff748	сохраненный ebp
6	804844b	адрес возврата
7	7	фактический аргумент
8	b7fccff4	содержимое фрейма вызывающей функции
9	8048470	

```asm
f:
   push    ebp
   mov     ebp, esp
   sub     esp, 40
   mov     eax, dword [ebp+8]
   mov     dword [ebp-20], 1
   mov     dword [ebp-16], 2
   mov     dword [ebp-12], 3
   mov     eax, dword [ebp-20+eax*4]
   mov     dword [esp], .LC0
   mov     dword [esp+4], eax
   call    printf
   leave
   ret
```

Переполнение буфера

```
void function(char *str) {
   char buffer[16];
   strcpy(buffer,str);
}

int main(void) {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
```

```
#include <stdlib.h>
int system(const char *command);
```

```
function:
   push    ebp
   mov     ebp, esp
   sub     esp, 40
   mov     eax, dword [ebp+8]
   mov     dword [esp+4], eax
   lea     eax, [ebp-24]
   mov     dword [esp], eax
   call    strcpy
   leave
   ret
```

Пусть в момент вызова адрес возврата был размещен в стеке по адресу `0xbffff72c`, адрес функции system – `0x80492b0`

28 байт с символом ‘A’

Немного поменяем содержимое переменной `large_string`

“AAA…AA**\xB0\x92\x04\x08**AAAA**\x38\xF7\xFF\xBF**/bin/sh**\x00**”

Способы защиты: Канарейка на стеке

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

int my_bad_function(char* d_msg)
{
    char what[100];
    strcpy(what, d_msg);
    printf("%s\n", what);
}
```

```
   ...                  ; пролог
   mov      eax, dword [gs:20]
   mov      dword [ebp-12], eax
   xor      eax, eax
   ...                  ; тело функции
   mov      edx, dword [ebp-12]
   xor      edx, dword [gs:20]
   je       .L3
   call     __stack_chk_fail
.L3:
   ...                  ; эпилог
```

В последних версиях компилятора gcc проверка включена по-умолчанию. Отключается опцией `-fno-stack-protector`

Способы защиты:
… и не только

- Неисполняемый стек и данные

```
char buffer[] = {...};

typedef void (* func)(void);

int main(int argc, char** argv) {
   func f = (func) buffer;
   f();
   return 0;
}
```

- Address Space Layout Randomization (ASLR)
- Безопасные библиотеки

```
errno_t strcpy_s( char *strDestination,
                  size_t numberOfElements,
                  const char *strSource );
```

_chk-версии некоторых стандартных функций

```
void f(int i) {
   int a[3] = {1, 2, 3};
   printf("%x\n", a[i]);
}
```

i	Вывод на экран
0	1
1	2
2	3
3	b7702030
4	8049ff4
5	bfbcb3b8
6	804846b
7	7
8	b781aff4
9	8048490

```
f:
   push    ebp
   mov     ebp, esp
   sub     esp, 40
   mov     eax, dword [ebp+8]
   mov     dword [ebp-20], 1
   mov     dword [ebp-16], 2
   mov     dword [ebp-12], 3
   mov     eax, dword [ebp-20+eax*4]
   mov     dword [esp+4], .LC0
   mov     dword [esp], 1
   mov     dword [esp+8], eax
   call    __printf_chk
   leave
   ret
```

GCC FORTIFY_SOURCE

```
gcc -v
...
gcc version 4.4.3
```

В последних версиях компилятора gcc включено по-умолчанию.

```
void foo(char *string) {
   char buf[20];
   strcpy(buf, string);
}
```

```
foo:
   push    ebp
   mov     ebp, esp
   sub     esp, 56
   mov     eax, dword [ebp+8]
   mov     dword [esp+4], eax
   lea     eax, [ebp-28]
   mov     dword [esp], eax
   call    strcpy
   leave
   ret
```

Отключается макросом -D_FORTIFY_SOURCE=0

```
foo:
   push    ebp
   mov     ebp, esp
   sub     esp, 56
   mov     eax, dword [gs:20]
   mov     dword [ebp-12], eax
   xor     eax, eax
   mov     eax, dword [ebp+8]
   mov     dword [esp+8], 20
   mov     dword [esp+4], eax
   lea     eax, [ebp-32]
   mov     dword [esp], eax
   call    __strcpy_chk
   mov     eax, dword [ebp-12]
   xor     eax, dword [gs:20]
   jne     .L5
   leave
   ret
.L5:
   call    __stack_chk_fail
```

stack-protector + FORTIFY_SOURCE

Сравнение строк специализированные команды

- CMPSB сравнение байт
- CMPSW сравнение 16-разрядных чисел
- CMPSD сравнение 32-разрядных чисел
- CLD/STD – очистить/установить флаг DF

```
Tmp = [ESI] – [EDI];
Установить статусные флаги согласно TMP;
If (DF == 0) {
   ESI += sizeof(операнд);
   EDI += sizeof(операнд);
} else { // DF == 1
   ESI -= sizeof(операнд);
   EDI -= sizeof(операнд);
}
```

Сравнение строк равного размера

```
%include 'io.inc'

BUFSIZE equ 32

section .data
 s1 db 'some text'
    times BUFSIZE-$+s1 db 0

 s2 db 'some text...'
    times BUFSIZE-$+s2 db 0

section .text
global CMAIN
```

```
CMAIN:
   push ebp
   mov ebp, esp
   sub esp, 12
   mov dword [esp], s1
   mov dword [esp + 4], s2
   mov dword [esp + 8], BUFSIZE-1
   call my_strncmp ; возвращаем 0,
                   ; если строки равны
                   ; иначе – номер байта
                   ; в котором встретилось
                   ; различие (считаем с 1)
   PRINT_DEC 4, eax
   NEWLINE
   xor eax, eax
   mov esp, ebp
   pop ebp
   ret
```

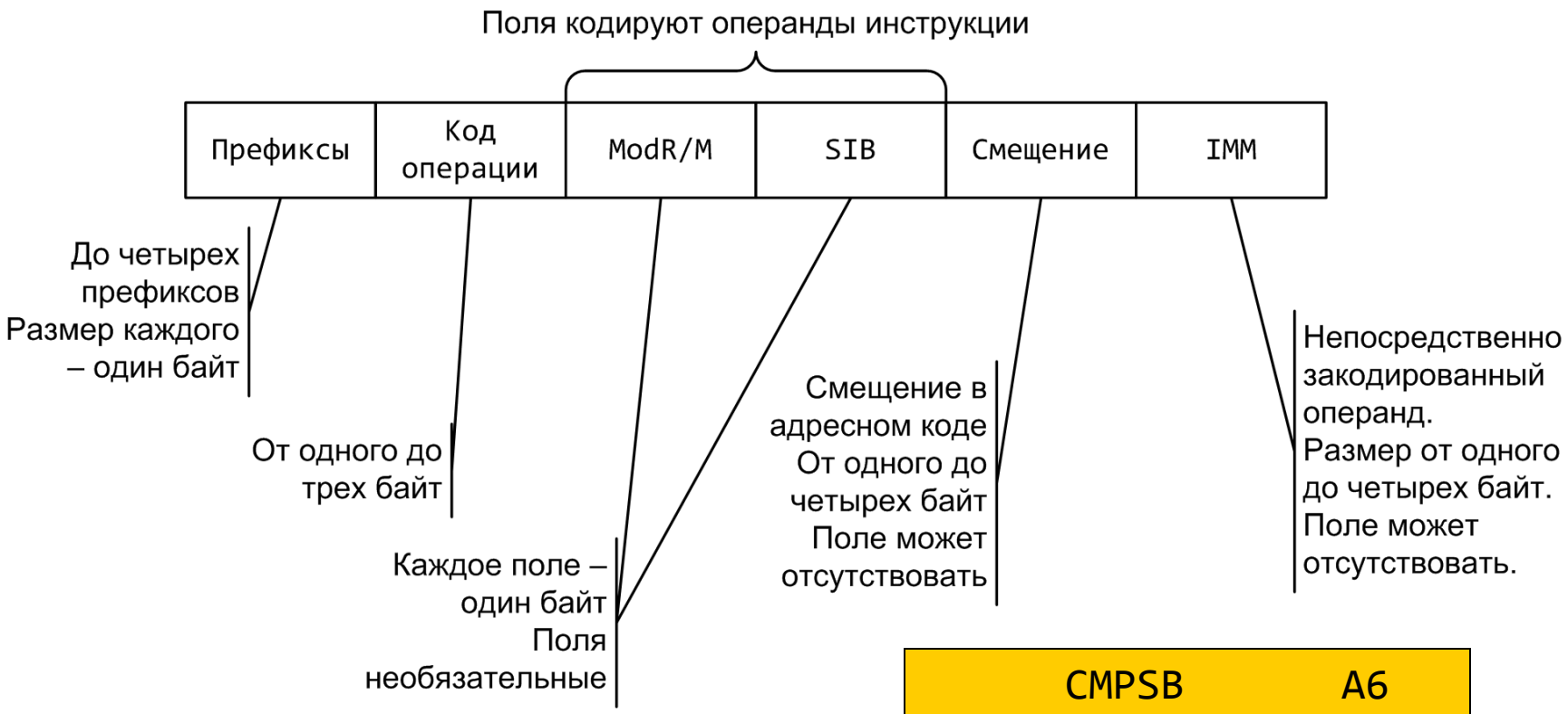
Сравнение строк равного размера

```
my_strncmp:
   push ebp
   mov ebp, esp
   push esi
   push edi
   xor eax, eax
   mov ecx, dword [ebp + 16]
   mov esi, dword [ebp + 8]
   mov edi, dword [ebp + 12]
   ;…


   ;…
.end:
   pop edi
   pop esi
   mov esp, ebp
   pop ebp
   ret
```

```
  ; …
  cld
  jecxz .end
.loop:
  cmpsb
  jne .ne
  loop .loop
  jmp .end
.ne:
  mov eax, dword [ebp + 16]
  sub eax, ecx
  inc eax
.end:
  ; …
```

Формат инструкции



	CMPSB		A6
REP	CMPSB	F3	A6
REPNE	CMPSB	F2	A6

Префикс повтора

- REPE = REPZ = REP
- REPNE = REPNZ

```
WHILE (ECX != 0) {
   Выполнить соответствующую строковую инструкцию;
   ECX = (ECX – 1); // данная операция на флаги не влияет
   IF (ECX == 0)
       break;
  IF (((Префикс повтора == REPZ или REPE) и
       (ZF == 0)) или
       ((Префикс повтора == REPNZ) и (ZF == 1) ))) {
     break;
  }
```

Сравнение строк равного размера

```
my_strncmp:
   push ebp
   mov ebp, esp
   push esi
   push edi
   xor eax, eax
   mov ecx, dword [ebp + 16]
   mov esi, dword [ebp + 8]
   mov edi, dword [ebp + 12]
   ;…


   ;…
.end:
   pop edi
   pop esi
   mov esp, ebp
   pop ebp
   ret
```

```
   ; …
   cld
   repe cmpsb
   je .end
   mov eax, 1
   mov ecx, -1
   cmovl eax, ecx
.end:
   ; …
```

Приводим поведение функции к стандартному виду

- меньше -1
- равно 0
- больше 1

Остальные строковые команды

- SCAS(B|W|D)
  - Сравниваем аккумулятор со строкой [EDI]
- MOVS(B|W|D)
  - Копируем строки [EDI] ⇦ [ESI]
- STOS(B|W|D)
  - Выгружаем аккумулятор по адресу [EDI]
- LODS(B|W|D)
  - Загружаем в аккумулятор из адреса [ESI]
  - Комбинировать с префиксом повтора не целесообразно

strnlen

```
#include <string.h>
size_t strnlen(const char *s, size_t maxlen);

strnlen:
   push ebp
   mov ebp, esp
   push edi
   xor eax, eax
   mov ecx, dword [ebp + 12]
   mov edi, dword [ebp + 8]
   repne scasb
   mov eax, dword [ebp +12]
   sub eax, ecx
   dec eax
   pop edi
   mov esp, ebp
   pop ebp
   ret
```

memset

```
#include <string.h>
void *memset(void *s, int c, size_t n);
```

```
memset:
   push ebp
   mov ebp, esp
   push edi
   mov ecx, dword [ebp + 16]
   mov esi, dword [ebp + 8]
   mov al,  byte  [ebp + 12]
   rep stosb
   pop edi
   mov esp, ebp
   pop ebp
   ret
```

Компактные функции стандартной библиотеки компилятор может встроить непосредственно в место вызова

Далее

- Представления для вещественных чисел
  - Дробные двоичные числа
  - Числа с плавающей точкой
- Сопроцессор x87
  - Устройство
  - Примеры программ