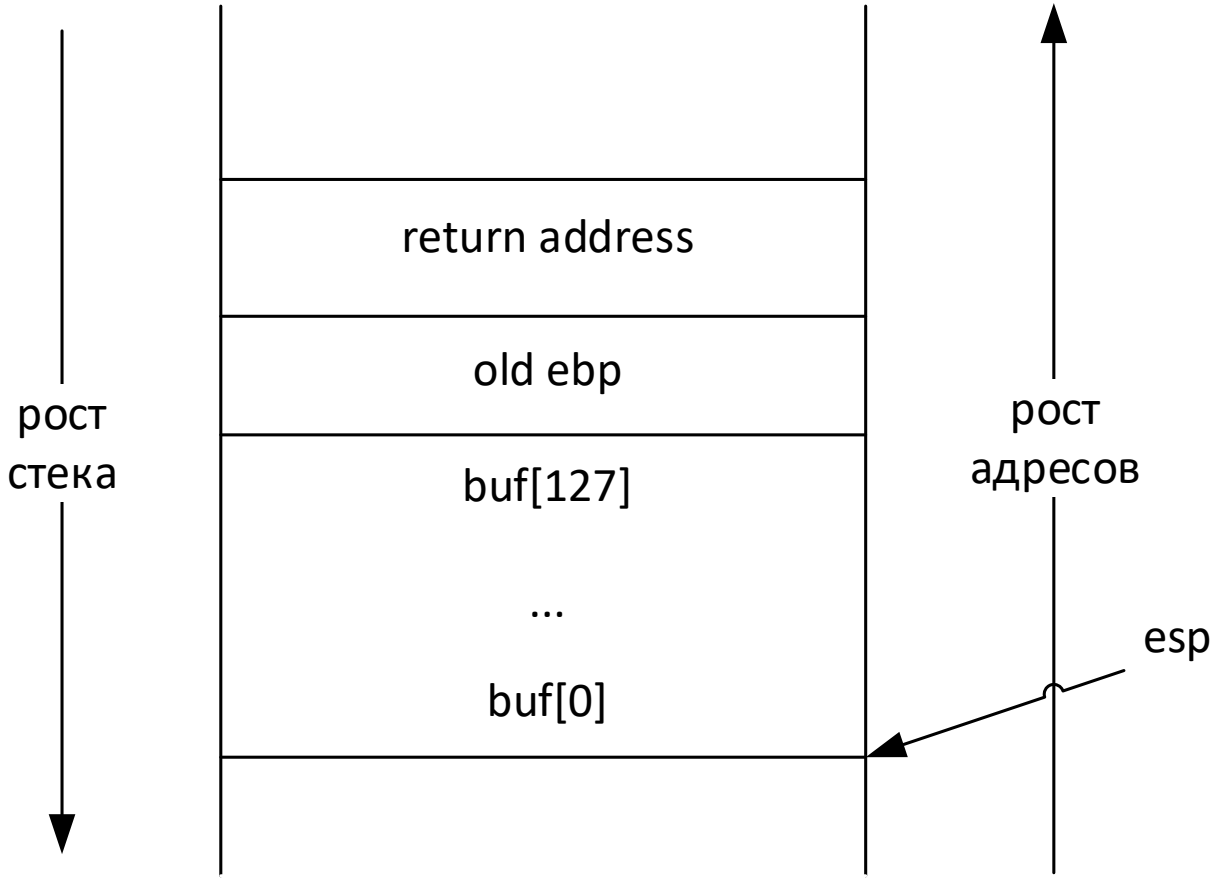
ОБРПО. Лекция 2 Переполнение буфера

ТЮРИН КАЙ АНДРЕЕВИЧ

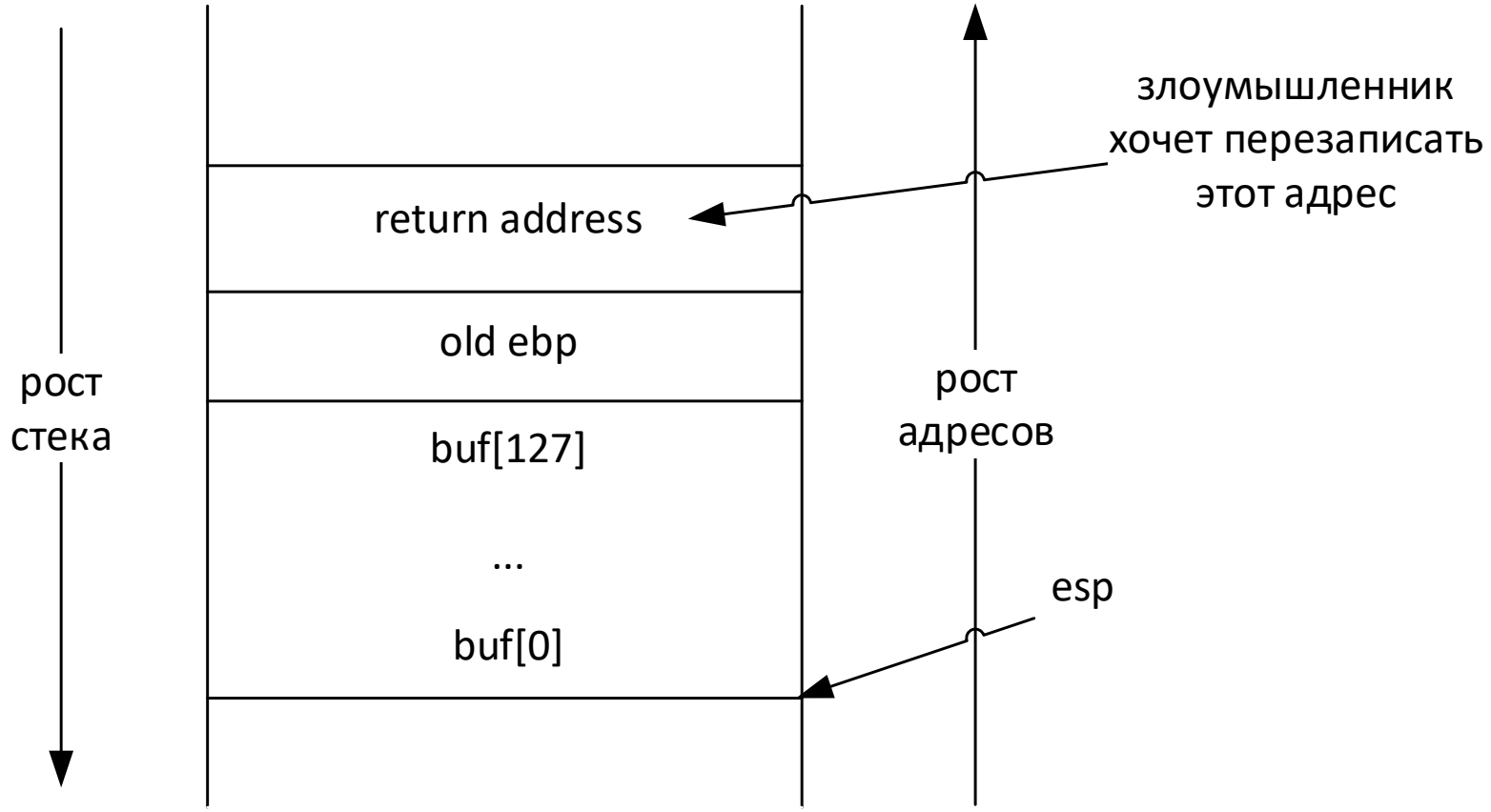
О чём мы сегодня говорим

Преимущественно язык C и платформа x86

Стек программы



Стек программы



Stack Overflow

vs.

Stack Buffer Overflow

ЗАЧЕМ???

LPE – local privilege escalation

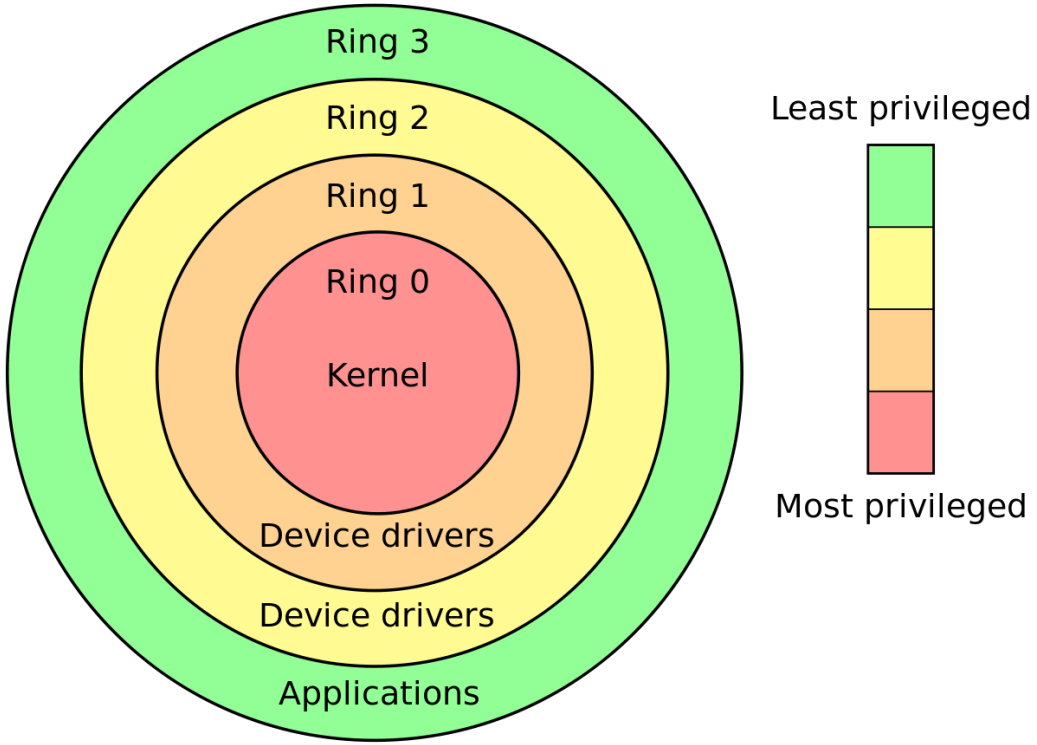
RCE – remote code execution

А почему ОС не препятствует?

ОС не «следит» за исполнением программы в каждый момент времени.

ОС на самом деле – набор сервисов для IO и взаимодействия с «железом».

Программы исполняются «свободно», но на другом уровне привилегий.



Способы защиты от переполнения буфера

Не писать баги!

gets – плохая идея

scanf – плохая идея

…

Да и вообще надо смотреть на warning’и компилятора

Использовать вспомогательные инструменты

1. Статический анализ

2. Фаззинг

Статический анализ

```
void foo(int* p){
    int off;
    int* z = p + off;
    . . .
}
```

Фаззинг

Тестирование программы случайными данными с максимизацией покрытых путей

```
void foo(int* p){
    int off;
    int* z = p + off;
    if (off > 8){
    . . .
    } else . . .
}
```

Разработка на безопасных языках

Java

C#

Python

…

Почему нет?

Большое количество легаси (нельзя переписать по техническим или бизнес-причинам)

Необходимость работы на низком уровне (с железом)

Требования к производительности (хотя есть JIT, WASM, … и вообще есть программы с большей нагрузкой на I/O)

В чем заключается эксплуатация?

1. Размещение кода (на самом деле часто не проблема)
2. Перехват управления и его передача на вредоносный код

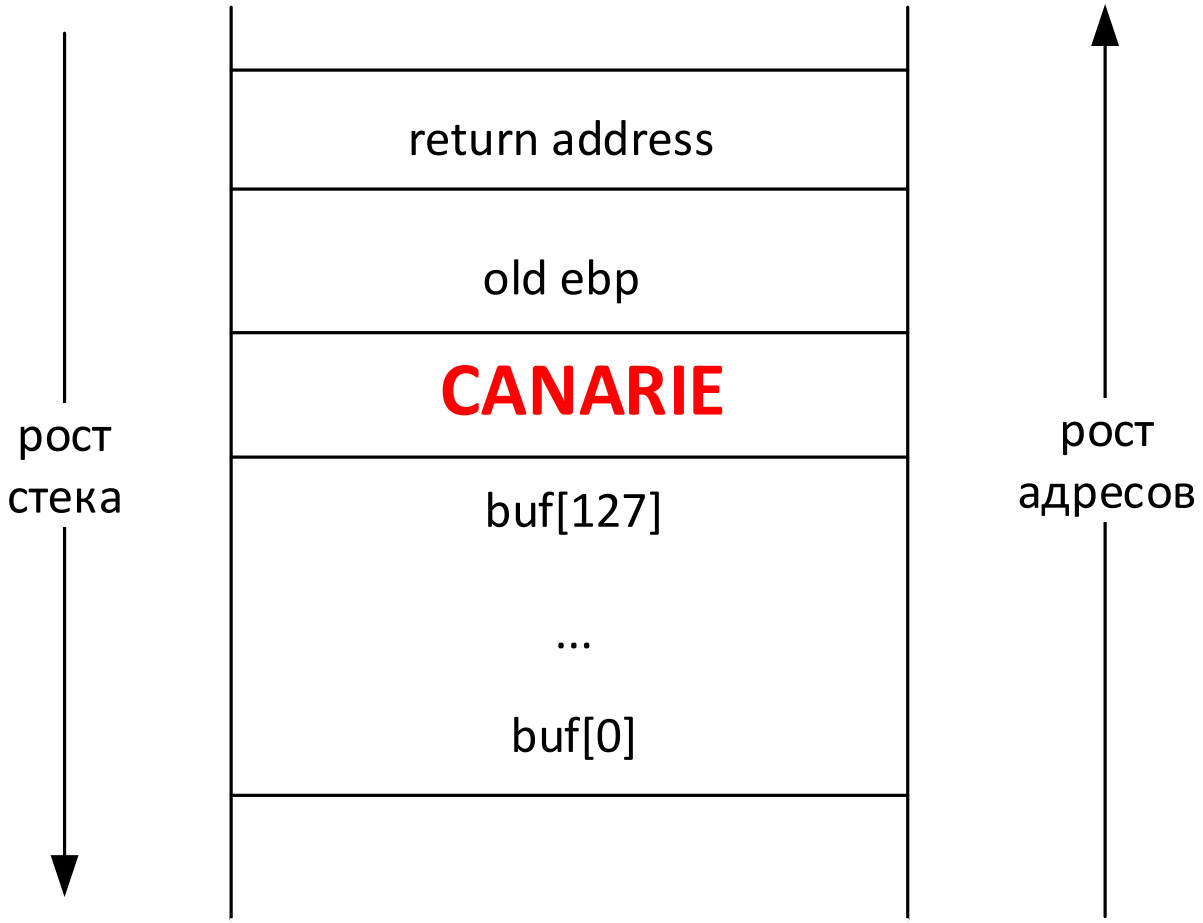
Есть много способов разместить код (или использовать существующий), поэтому сосредоточимся на перехвате управления.

Stack canaries

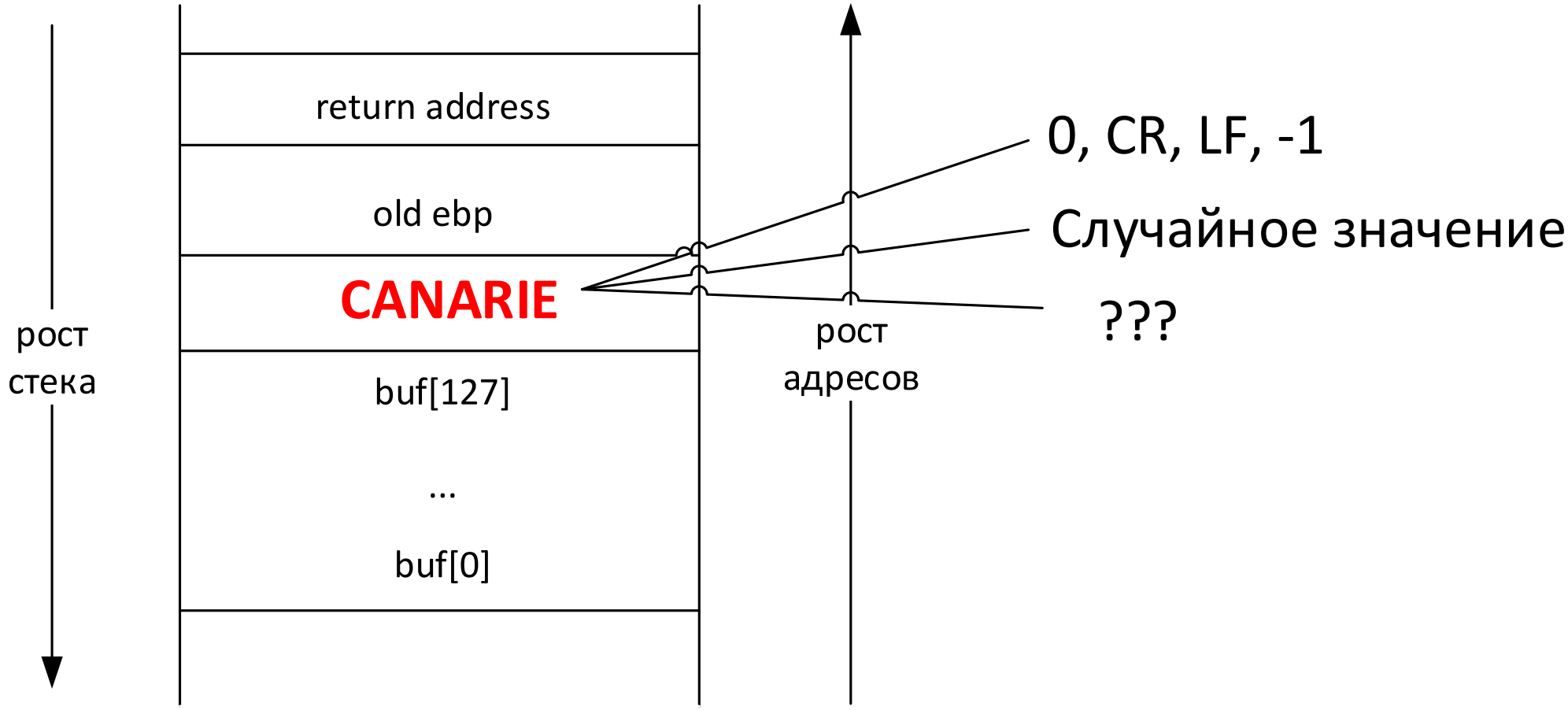
Допустить переполнение буфера – ОК.

Но нужно его обнаружить и предотвратить передачу управления вредоносному коду!

Stack canaries



Stack canaries



```
int read_req(void){
        char buf[128];
        int i;
        scanf("%s", buf);
        i = atoi(buf);
        return i;
}
```

Компиляция кода без канарейки

Windows:

cl main.c /Zi /GS-

Linux:

gcc main.c -fno-stack-protection -o main

```
; int __cdecl read_req()
read_req        proc near               ; CODE XREF: j_read_req↑j
                                        ; DATA XREF: .pdata:000000014008E03C↓o

var_98          = dword ptr -98h
string          = byte ptr -88h

                sub     rsp, 0B8h
                lea     rdx, [rsp+0B8h+string]
                lea     rcx, _Format    ; "%s"
                call    j_scanf
                lea     rcx, [rsp+0B8h+string] ; string
                call    j_atoi
                mov     [rsp+0B8h+var_98], eax
                mov     eax, [rsp+0B8h+var_98]
                add     rsp, 0B8h
                retn
read_req        endp
```

Компиляция кода с канарейкой

Windows:

cl main.c /Zi

Linux:

gcc main.c -o main

```
; int __cdecl read_req()
read_req        proc near               ; CODE XREF: j_read_re
                                        ; DATA XREF: .pdata:0(

var_A8          = dword ptr -0A8h
string          = byte ptr -98h
var_18          = qword ptr -18h

                sub     rsp, 0C8h
                mov     rax, cs:__security_cookie
                xor     rax, rsp
                mov     [rsp+0C8h+var_18], rax
                lea     rdx, [rsp+0C8h+string]
                lea     rcx, _Format    ; "%s"
                call    j_scanf
                lea     rcx, [rsp+0C8h+string] ; string
                call    j_atoi
                mov     [rsp+0C8h+var_A8], eax
                mov     eax, [rsp+0C8h+var_A8]
                mov     rcx, [rsp+0C8h+var_18]
                xor     rcx, rsp        ; StackCookie
                call    j___security_check_cookie
                add     rsp, 0C8h
                retn
read_req        endp
```

```
; uintptr_t _security_cookie
__security_cookie dq 2B992DDFA232h
```

```
; void __cdecl _security_check_cookie(uintptr_t StackCookie)
__security_check_cookie proc near       ; CODE XREF: j___security_check_cookie↑j
                                        ; DATA XREF: .pdata:0000000140092548↓o
                cmp     rcx, cs:__security_cookie
                repne jnz short ReportFailure
                rol     rcx, 10h
                test    cx, 0FFFFh
                repne jnz short RestoreRcx
                repne retn
; ---------------------------------------------------------------------------

RestoreRcx:                             ; CODE XREF: __security_check_cookie+13↑j
                ror     rcx, 10h        ; stack_cookie

ReportFailure:                          ; CODE XREF: __security_check_cookie+7↑j
                jmp     j___report_gsfailure
__security_check_cookie endp
```

Когда канарейки не спасают?

Перезапись указателей

```
int* ptr = . . .;
char buf[128];
gets(buf);
*ptr = 5;
```

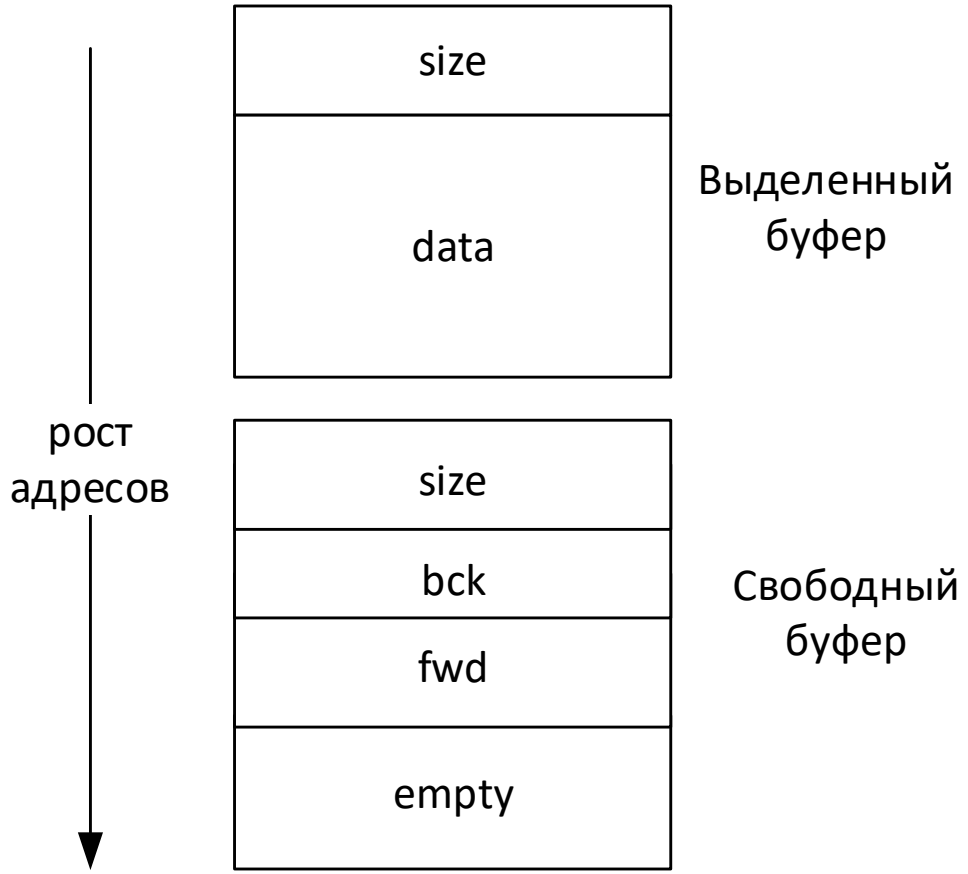
Когда значение канарейки можно угадать

Значение не всегда настолько случайно, насколько нам хотелось бы!

malloc/free

```
char* p, q;
p = malloc(1024);
q = malloc(1024);
strcpy(p, buf);
free(p);
free(q);
```

Расположение буферов в памяти



Код слияния страниц

```
p = get_free_block_struct(size);
bck = p->bk;
fwd = p->fd;
fwd->bk = bck;
bck->fd = fwd;
```

Литература

MIT 6.858 Computer Systems Security

Хакинг: искусство эксплойта

http://kmb.ufoctf.ru/

Документация к любимому компилятору

RE4B

Practical Reverse Engineering: X86, X64, ARM, Windows Kernel, Reversing Tools, and Obfuscation