# Segurança da Computação

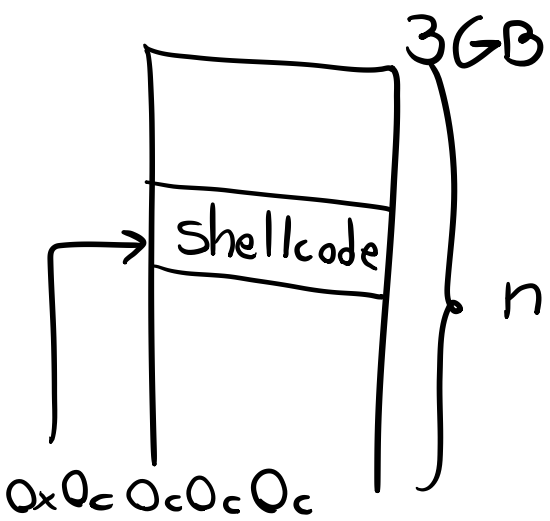
Matheus Venturyne Xavier Ferreira

Universidade Federal de Itajubá

21 de Outubro de 2015

# Ataque a ASLR + DEP (Data Execution Prevention)

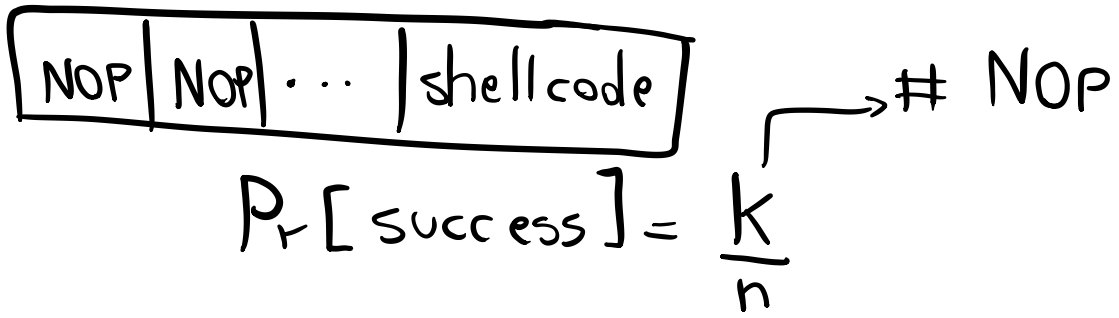
- Ataque probabilístico




$$P[\text{jump shellcode } 0x0c\ldots] = \frac{1}{n}$$

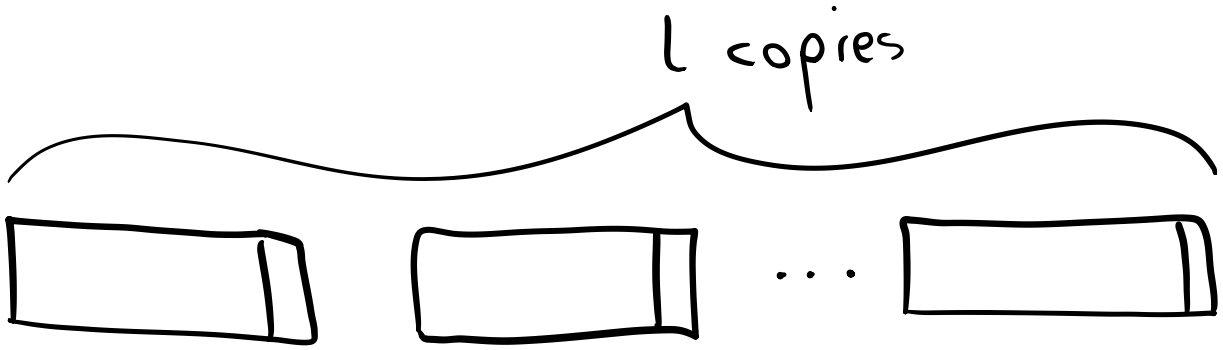
# Ataque a ASLR + DEP – Melhora 1

- NOP (0x90) sleds



# Ataque a ASLR + DEP – Melhora 2

- Muitas cópias de NOPs + Shellcode




$$Pr[success] = \frac{kl}{n}$$ (8% chance of success ⇒ good)

# Browsers

- Browsers rodam JavaScript (JS) que permitem a execução de dados e são vulneráveis a ataques de HeapSpray
- Deve se preocupar com Unicode

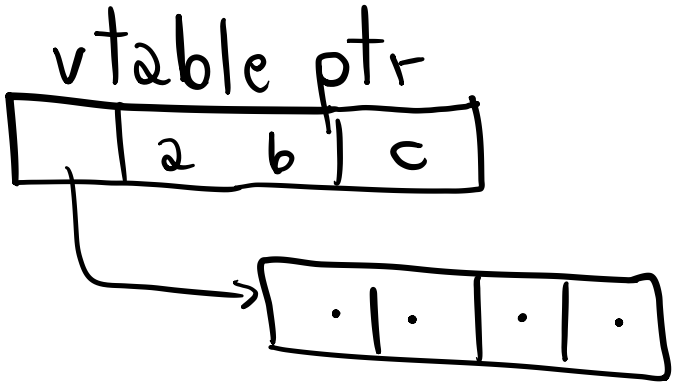
```
Shellcode = unescape("_____")
Nop = unescape("%u0c0c%u0c0c")
For(sled="";sled.length < 0x10000; sled+=nop)
Spray = new Array()
For(I = 0; I < 5000; I++)
        spray[I] = sled + shellcode
```

# Browsers

- Como o browser implementa esse código
- JavaScript não possui a noção de heap e stack
- Browsers devem implementar a noção de heap e stack para rodar JavaScript
- Browsers podem detectar que estão tentando colocar várias cópias de algo (shellcode) na memória
  - Firefox não irá encher a memória. Iria fazer apenas uma cópia e dizer que os ponteiros apontam para a mesma coisa

# Métodos Virtuais

- Implementado com vtables
- Ataques de HeapSpray normalmente direcionam para o endereço 0x0c0c0c0c pois é onde se encontram as vtables



# DEP

- Uma página na memória não pode ser executável ou passível de ser escrita
- JITs são uma exceção (JavaScript)
  - Uma forma de atacar um Iphone
- Mas porque é seguro executar javascript?
  - Há limites com o que se pode fazer com JavaScript
- Porque são úteis?
  - Se executados em ordem

# Ataques com JavaScript

- JavaScript pode acessar cookies dentro do mesmo dominio mesmo que as aplicações sejam diferentes
  - www.google.com/{maps,mail}
- History sniffing: permite que JavaScript minere os websites que você já esteve antes
- Taint: permite rastrear onde alguns cookies ou informações especiais vão
- Solução:
  - Sempre retornar blue color quando perguntado por JavaScript
- Contra-ataque: enviar um pedido para um website e medir o tempo para carregar. Se carregou rápido, a página estava na cache; portanto, ela foi acessada antes

# Double Free

- Malloc é utilizado para alocar memória dinámica em tempo de execução.
  Toda memória alocada é dealocada em algum momento da execução do programa
- Memória dinâmica é armazenada na Heap
  - Uma estrutura de dados implementadas em C. Não necessariamente possui suporte do Sistema operacional.
- Algumas implementações de Malloc são vulneráveis a memória que é liberada duas vezes

# Heap – Inspirado por K&R2 malloc() e Doug Lea malloc()

```
/*
 * the chunk header
 */
typedef double ALIGN;

typedef union CHUNK_TAG
{
  struct
    {
      union CHUNK_TAG *l;    /* leftward chunk */
      union CHUNK_TAG *r;    /* rightward chunk + free bit (see below) */
    } s;
  ALIGN x;
} CHUNK;
```

# Heap – Inspirado por K&R2 malloc() e Doug Lea malloc()

```
/*
 * we store the freebit -- 1 if the chunk is free, 0 if it is busy --
 * in the low-order bit of the chunk's r pointer.
 */

/* *& indirection because a cast isn't an lvalue and gcc 4 complains */
#define SET_FREEBIT(chunk) ( *(unsigned *)&(chunk)->s.r |=  0x1 )
#define CLR_FREEBIT(chunk) ( *(unsigned *)&(chunk)->s.r &= ~0x1 )
#define GET_FREEBIT(chunk) ( (unsigned)(chunk)->s.r & 0x1 )
```

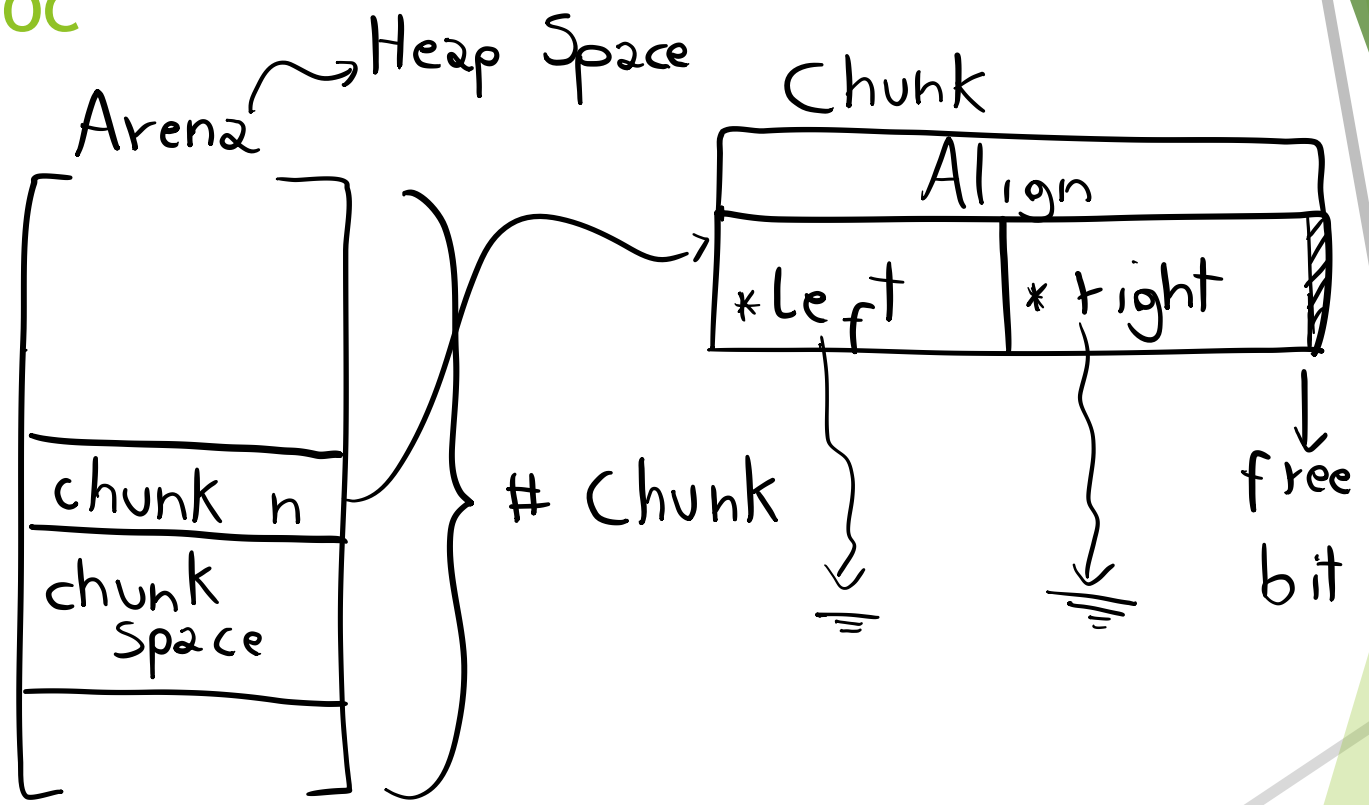
# Heap – Inspirado por K&R2 malloc() e Doug Lea malloc()

```
/* it's only safe to operate on chunk->s.r if we know freebit
 * is unset; otherwise, we use ... */
#define RIGHT(chunk) ((CHUNK *)(~0x1 & (unsigned)(chunk)->s.r))

/*
 * chunk size is implicit from l-r
 */
#define CHUNKSIZE(chunk) ((unsigned)RIGHT((chunk)) - (unsigned)(chunk))

/*
 * back or forward chunk header
 */
#define TOCHUNK(vp) (-1 + (CHUNK *)(vp))
#define FROMCHUNK(chunk) ((void *)(1 + (chunk)))
```

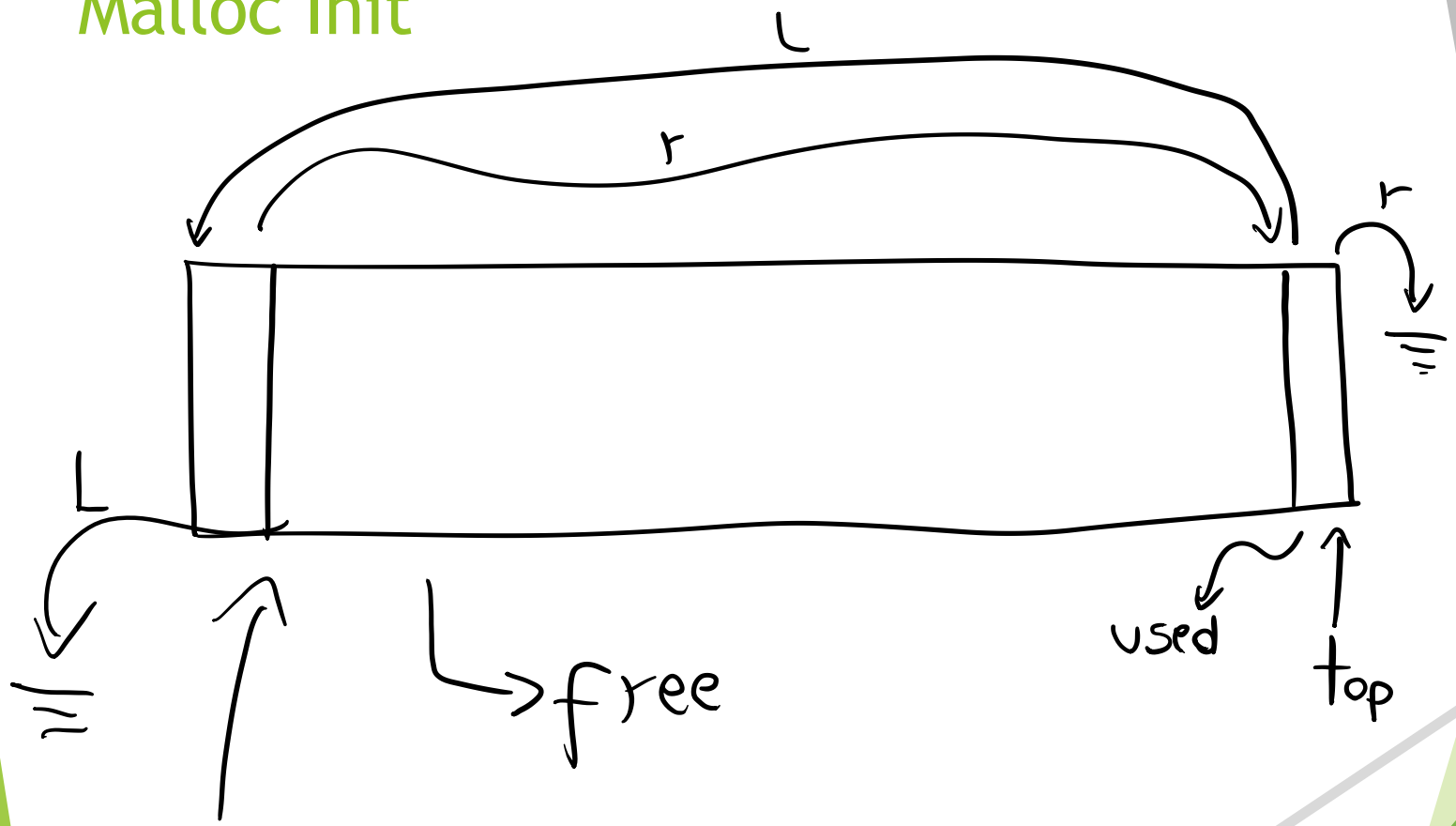
# Heap – Inspirado por K&R2 malloc() e Doug Lea malloc()

```
/* for demo purposes, a static arena is good enough. */
#define ARENA_CHUNKS (65536/sizeof(CHUNK))
static CHUNK arena[ARENA_CHUNKS];

static CHUNK *bot = NULL;   /* all free space, initially */
static CHUNK *top = NULL;   /* delimiter chunk for top of arena */
```

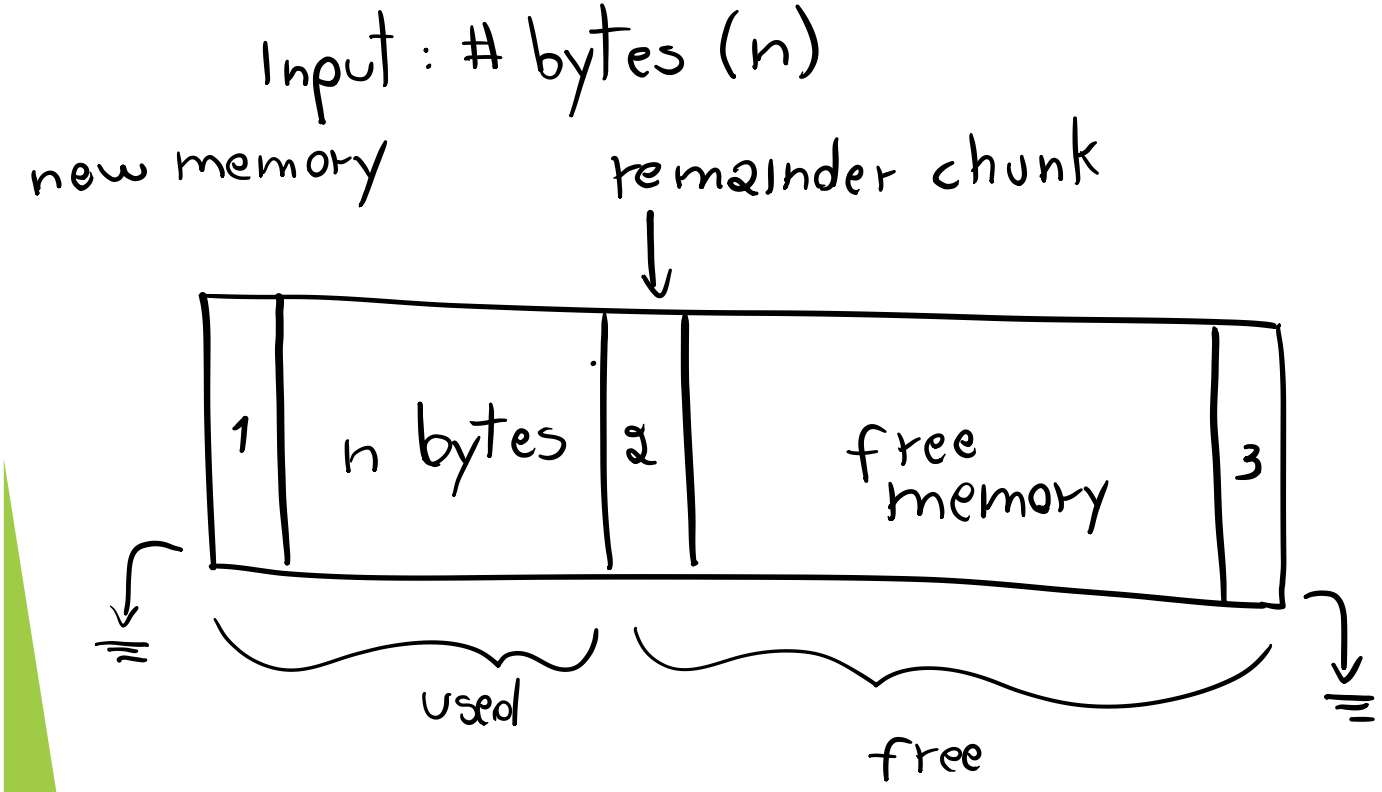
# Malloc



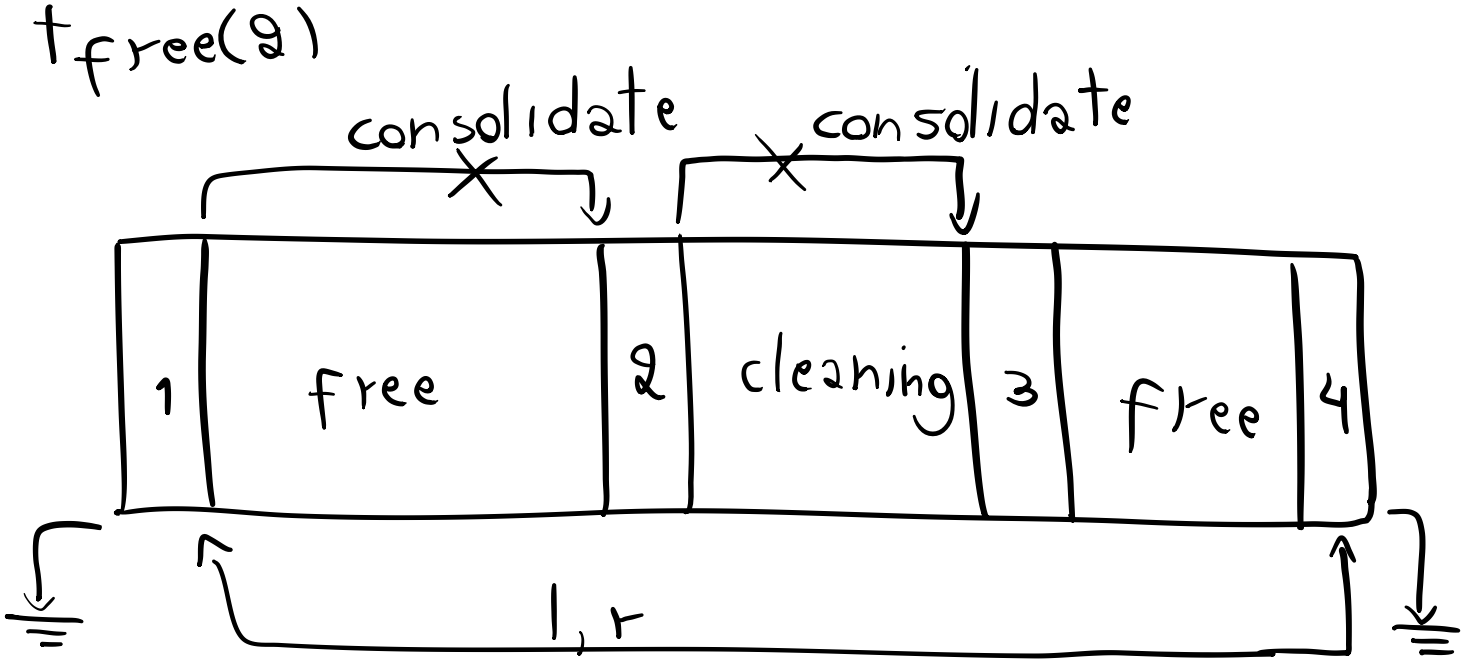
# Malloc Init



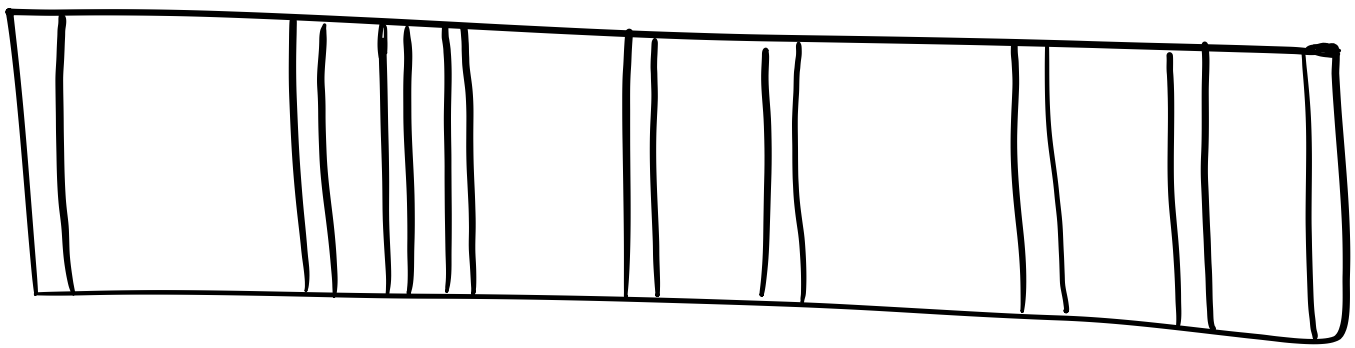
# Malloc tmalloc



# Malloc tfree



# Heap Fragmentation



# Referências


- M. Zalewski: Browser Security Handbook, chapters 1 (basic concepts) and 2 (standard security features).
- D. Blazakis: Interpreter Exploitation
- Anonymous, “Once Upon a free()...,” Phrack 57 #0x09.
- Anonymous, “Bypassing PaX ASLR Protection,” Phrack 59 #0x09.