# Memory Eye


**SSTIC 2011**

Yoann Guillot
Sogeti / ESEC R&D
yoann.guillot(at)sogeti.com

# Plan

# Memory Eye

- Analyse globale d'un programme
- Un outil pour analyser la zone de mémoire dynamique d'un programme
- Analyse de patterns

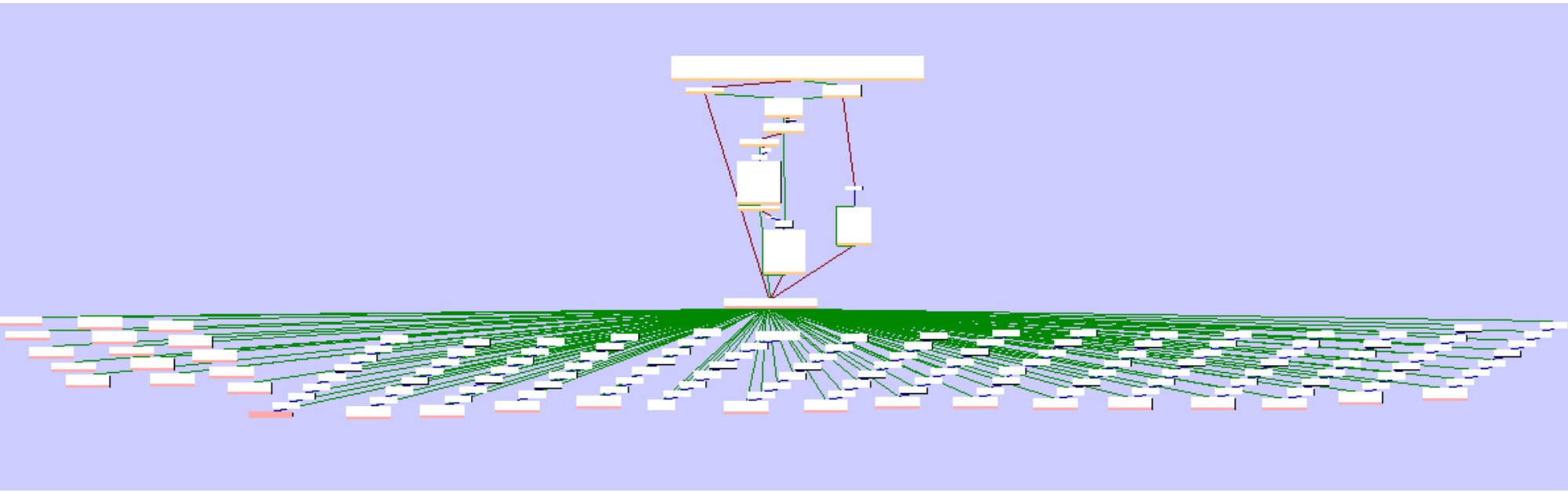
# Pourquoi ?

L'assembleur, c'est compliqué

- Instructions bizarres
  - xor
  - pmulhrsw
- Architectures plus ou moins familières
  - x64
  - arm
- Algorithmes bizarres [1]

1. http://article.gmane.org/gmane.comp.lib.glibc.alpha/15278

# wtf.asm



# L'analyse de code

## Pros

- Bons outils
  - IDA
  - BinNavi [a]
- Grosse communauté

a. M.I.A 01/03/11

## Cons

- Approche bottom-up
- Changement de version de la cible == reset
  - BinDiff
- prison

# Plan

# Taxinomie

## Pile

- Limité a une fonction (+ sous-fonctions)
- Taille contrainte

## Globales

- Fixé à la compilation
- Durée de vie : permanente
- Inclues dans le binaire [a]
- Rigide

a. sauf .bss

# Taxinomie (2)

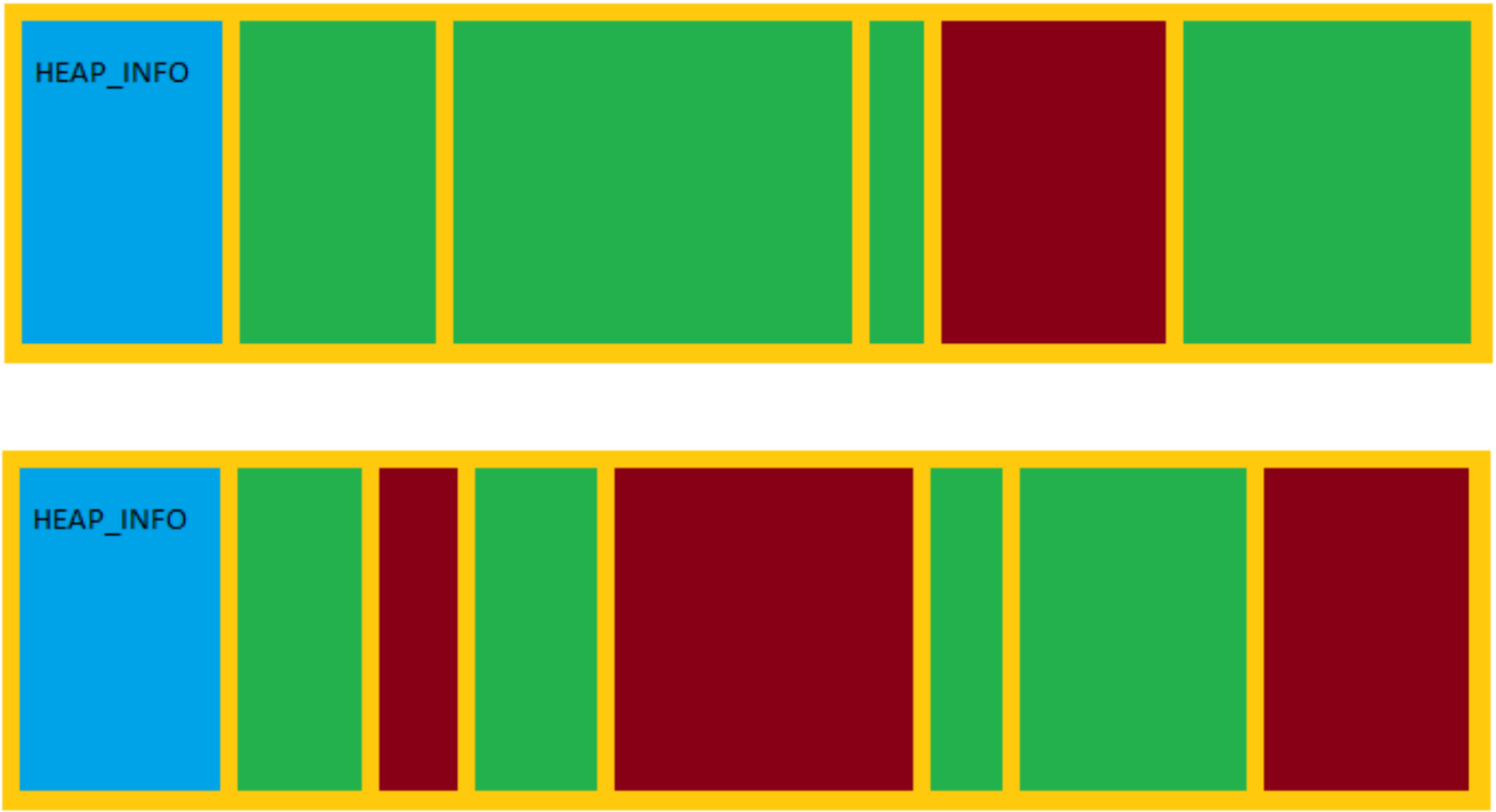
## Tas (*Heap*)

- Dynamique
  - Allocation/Déallocation à la demande
  - Utilisation optimale de la RAM
- Durée de vie personnalisée

# Structure du tas

- "Tas" = 1+ zones independantes
  - Par librairie
  - Par thread
- Chaque est une succession de *chunks*
- Chunk = header + data
- Gestion des chunks via *malloc()* et *free()*
- Optimisations (lookaside)

# Heap



# Heap

# Allocateur personnalisé

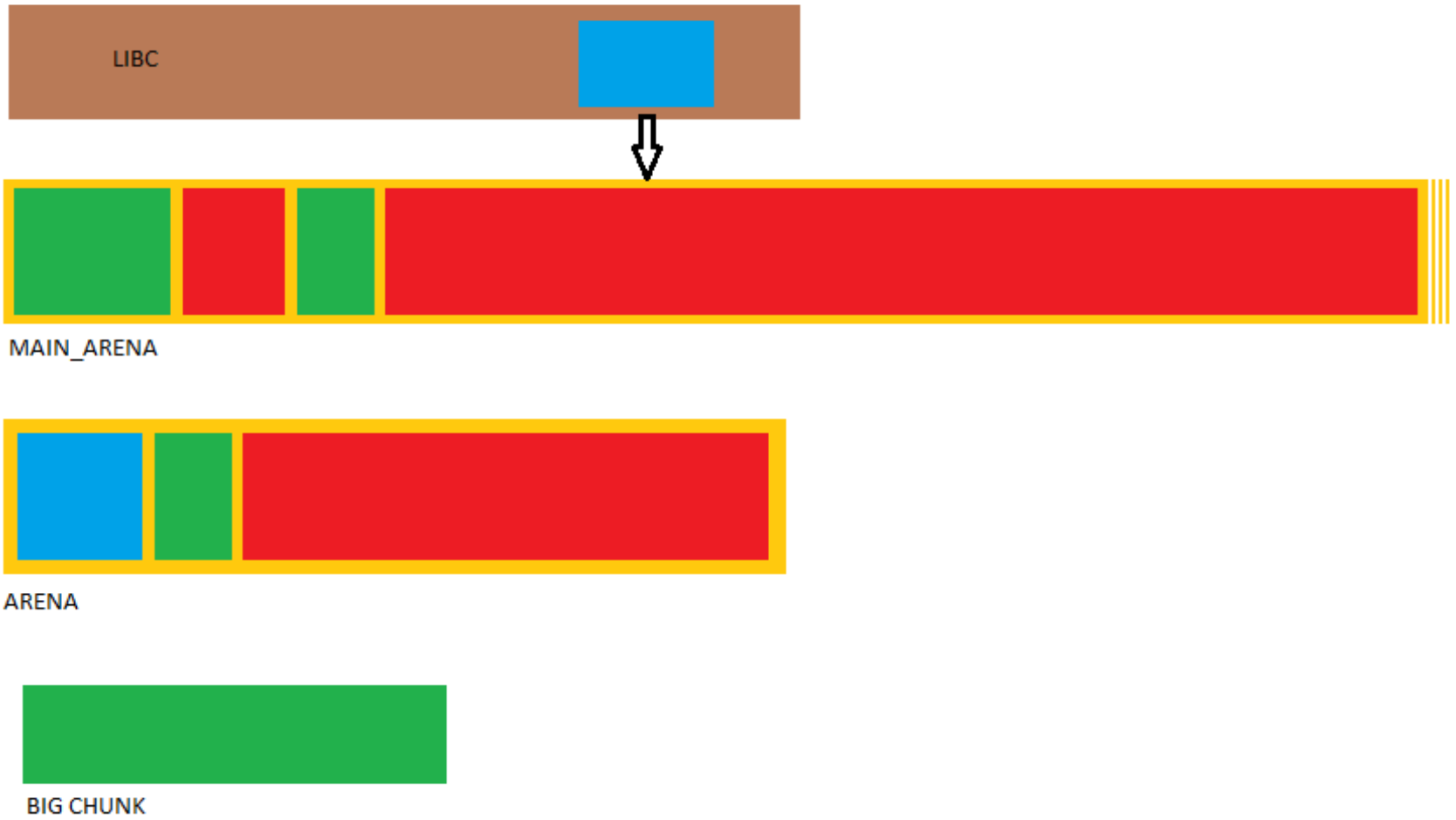
- OS = Allocateur générique
- Application = Allocateur(s) personnalisé
  - spécificités de taille de chunks
  - gestion de la fragmentation
  - rapidité

# Plan

# GLibc 2.11

- Zone independante = “arena”
- Zone initiale **main_arena** = *sbrk()*
- Zones secondaires = *mmap()*
- Gros chunks = *mmap()*
- Cache de chunks libre = *lookaside*

# Heap Linux



# GLibc : extraction du main_arena

L'adresse du **main_arena** n'est pas disponible.

## Approche 1

- Ignorer le problème
- Sbrk depuis /proc/*pid*/maps (`[heap]`)
- PB : pas d'arenas secondaires, pas de lookaside

## Approche 2

1. Trouver une fonction qui réference la structure
2. En extraire l'adresse
3. Liste chaînée des arenas secondaires

Dans tous les cas, il faut scanner les chunks *mmap()*

# GLibc : extraction du main_arena

```
dasm.disassemble_fast('malloc_trim')
b = dasm.block_at('malloc_trim')
if b.list.last.opcode.name == 'call'
  # x86 getip()
  dasm.disassemble b.to_normal.first
end
# mutex_lock(&main_arena.mutex) gives us the addr
cmpxchg = dasm.decoded.values.find { |di|
  di.kind_of? DecodedInstruction and
  di.opcode.name == 'cmpxchg'
}
raise 'no_cmpxchg' if not cmpxchg
indir = cmpxchg.instruction.args.first.symbolic
arena_ptr = d.backtrace(indir.pointer, cmpxchg.address)
if arena_ptr.length == 1
        arena_ptr = arena_ptr.first.reduce
end
raise "cant_find_mainarena" if not arena_ptr.kind_of? Integer
arena_ptr += libc_base
```

# Enumeration des chunks

- Arena $\Rightarrow$ top chunk + len - system_mem
- Enumérer
- Soustraire le lookaside
- Scanner les mmap chunks

# Plan

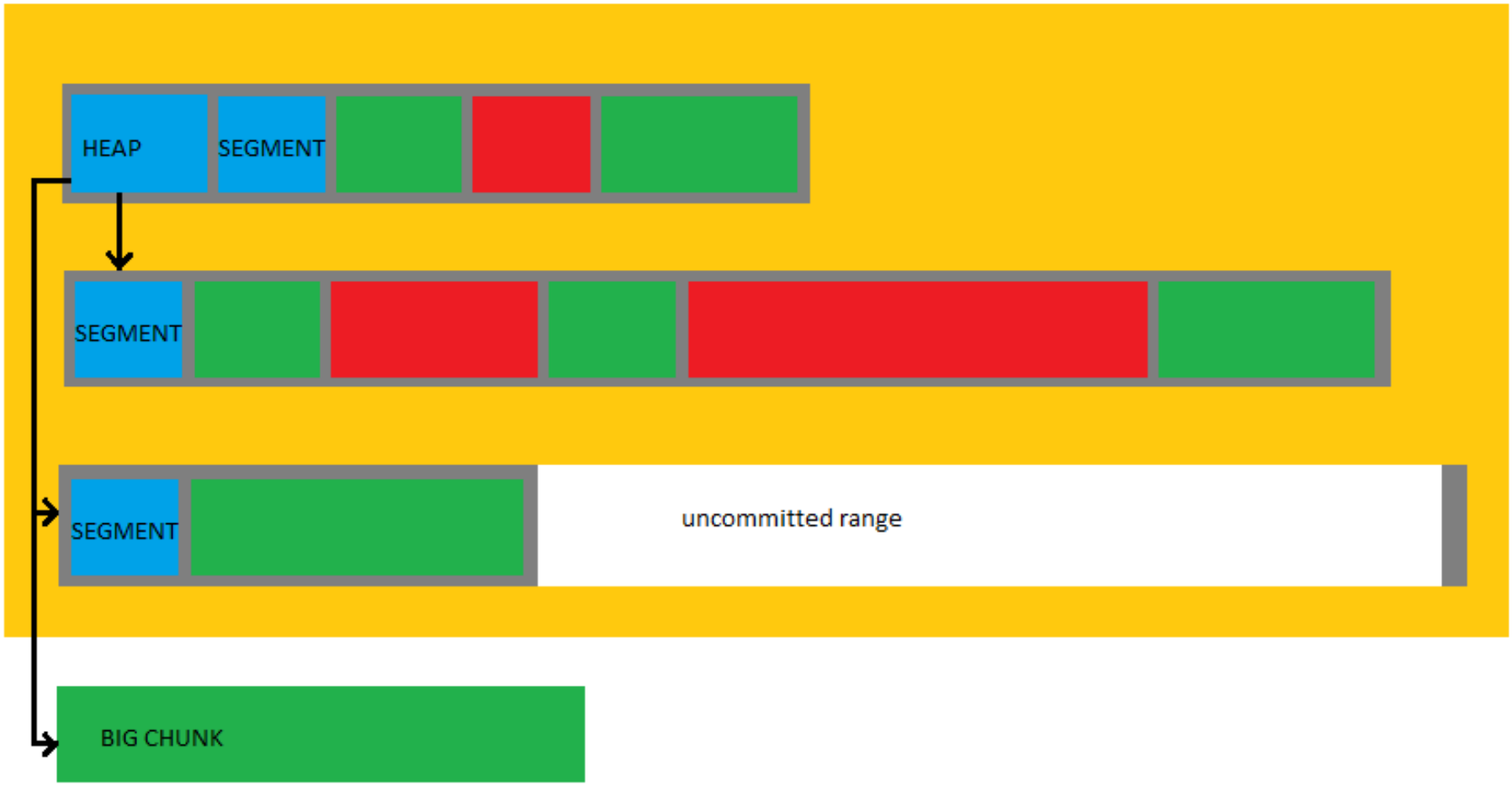
# Windows XP

- Zone independante = "_HEAP"
- Gros chunks = *VirtualAlloc()*
- Cache de chunks libre = *LookAside*
- *ToolhelpSnapshot(HEAPLIST)*

# Windows heap

- Heaps divisés en Segments
- Taille d'un segment = double du précedent
  - FirstEntry
  - LastEntryInSegment
  - UncommittedRange

# Heap Windows



# Windows 7

- Remaniement de **_HEAP**
- Suppression du LookAside
- Remplacé par le Low Fragmentation Heap

# Plan

# Et après ?

Construction du graphe de cross-references

- Scanne le contenu de chaque *chunk*
- Pointeur vers *chunk* $\Rightarrow$ arête du graphe

# Classes d'équivalences

- Tableaux
  - 1 *chunk* pointant sur de nombreux *chunks* de même taille
- Listes chaînées
  - *chunk* pointant sur un autre de même taille
  - le suivant fait pareil
  - le pointeur est au même offset

# Plan

# Dwarf Fortress

- `http://www.bay12games.com/dwarves/`
- Jeu massivement unijoueur
- Simulation d'un univers fantasy
- Communauté de hackers active
- Univers extrêmement détaillé
- Code C++
- Mis à jour régulièrement

$\Rightarrow$ Application idéale

# Démo

Démo

# Autres applications

- Tout programme manipulant des données bien différenciées

# Questions ?

# Références


- http://www.blackhat.com/presentations/bh-usa-09/MCDONALD/BHUSA09-McDonald-WindowsHeap-PAPER.pdf
- http://lateralsecurity.com/downloads/hawkes_ruxcon-nov-2008.pdf
- http://www.eglibc.org/cgi-bin/viewcvs.cgi/branches/eglibc-2_13/libc/malloc/malloc.c?rev=12752&view=markup
- http://metasm.cr0.org/