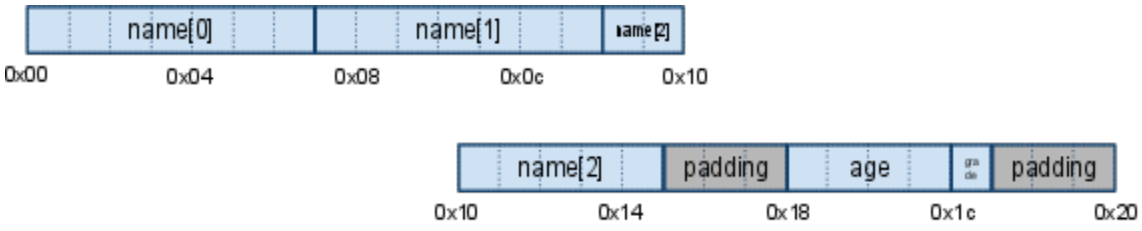
CS61: Systems Programing and Machine Organization

Fall 2011

Week 3: Monday 26 September – Friday 30 September

Solutions

1. The following diagram shows the layout of a struct student_t. Hex numbers indicate the offset from the beginning of the struct. The struct is represented using 32 contiguous bytes. For reasons of space, we draw this over two lines.
   Note that the field age is aligned at a multiple of four bytes from the beginning of the struct, and that padding is added at the end of the struct to make the total size a multiple of 4. These two facts ensure that in an array of struct student_t, the age fields are always placed at a memory address that is divisible by 4, as per the Linux/x86 alignment requirements.

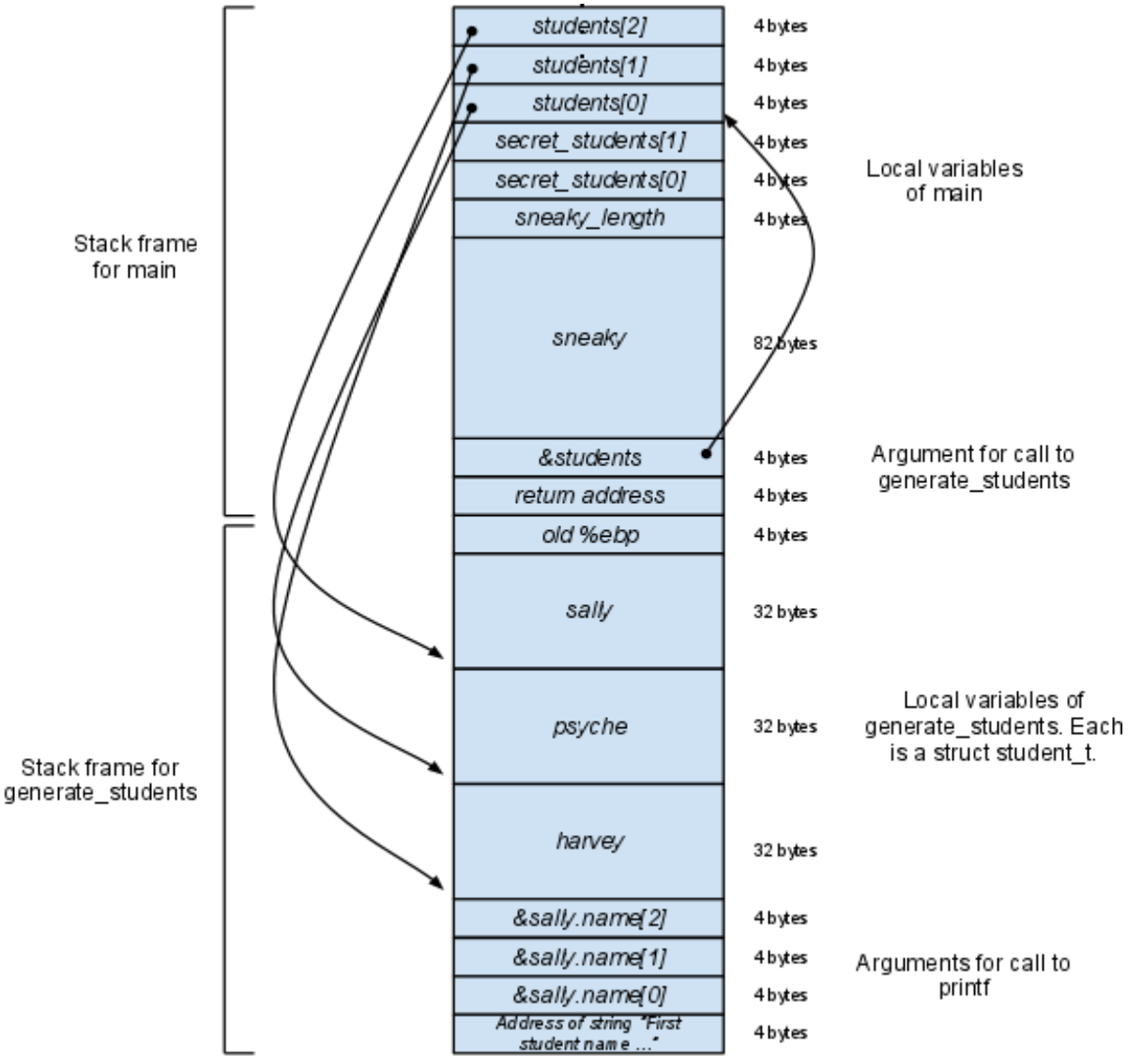


2.
   a. 0x80049110 + 0x0 = 0x80049110
   b. 0x80049110 + 0x0 = 0x80049110
   c. 0x80049110 + 0x7 = 0x80049117
   d. 0x80049110 + 0x1c = 0x8004912c
   e. 0x80049110 + 0x20 + 0x1c = 0x8004914c
3. “Sally HenthorRo Ro” because “Henthorn” plus the null terminating character at the end is 9 characters. Thus, only the first 7 characters will fit in the name[1] field. “Ro” will overwrite the last two characters of “Henthorn”, and when the system attempts to print the name[1] field, the first null terminating character it encounters is after “Ro,” so it prints the first 7 characters of “Henthorn” followed by “Ro.”
4. The following diagram shows the state of the stack at the end of the function generate_students, that is, just after the return from printf. We assume that the function

main does not push any caller-save registers on to the stack, and that generate_students does not push any callee-save registers on to the stack. We do not show the stack above the local storage for main, or below the stack from for generate_students. (What will the memory addresses below the stack frame for generate_students contain?) We also assume that the compiler is being fairly dumb about optimizing the code. For example, since the value of the variable sneaky_length is constant, a compiler might replace all uses of the variable with its constant value (82), and thus not need to compute the value of sneaky_length at runtime, nor store it on the stack.

The diagram is not drawn to scale, but the size of each element on the stack is indicated on the right of the diagram.

The arrows indicate what locations point to what other location. For example, in the location students[0] is the address of the struct student_t psyche. Similarly, students[1] contains the address of harvey, and the argument of the call to generate_students is the address of the array students (which is equal to the address of the first element of the array, students[0]). (Due to lack of space in the diagram, we do not show arrows from the arguments of the call to printf into the struct student_t sally.)



5. See answer for #4
6. See answer for #4
7. Technically, we don't actually know what it will print. It depends on what printf() allocates on the stack. Most likely, it will print "Secret Betty Crock".
   The pointers inserted into "students" pointed to the stack frame for generate_students(). When generate_students() returns, the stack frame is de-allocated.

   When generate_secret_students() is called, its stack frame starts at the same place the stack frame generate_students() used to start.

   Thus, where generate_students() allocated the struct on the stack for sally, generate_secret_students() allocates a struct for oscar meyer.

   Then, where generate_students() allocated the struct for psyche, generate_secret_students() allocates space for the betty crock struct. Since the first pointer in students has been set to point to the space where psyche's struct resided, and the betty crock struct is now located there, students[0] dereferences to the betty crock struct, and thus "Secret Betty Crock" is printed.

   Note that when printf() is called, the stack frame for generate_secret_students() has already been deallocated as well, so technically this memory access is invalid.

8. Lines 60-67 are crafting a buffer overflow attack on generate_secret_students(). They are attempting to craft a buffer such that, when generate_secret_students() is called, the return address will be overwritten with 0xdeadbeef.

9. In order to reach the location of the return address in the previous stack frame, the buffer must skip over: the 9 byte character buffer and the two struct student_t's allocated at the beginning of the function, the return stack frame pointer (4 bytes), the return address itself (4 bytes), and space for a final null terminating character to make sure the strcpy() returns.