CS-E4760: Platform Security (2023)
Exercise 4: Memory protection

1. Heap memory management [4 points]

a. Data allocated on the heap should needs to be released when it is no longer used. Some common approaches are manual allocation, reference counting, and garbage collection.

   (i) With manual memory allocation (e.g. with malloc() and free()), the programmer must free an object when it is no longer in use. With reference counting, an object is freed when its reference count reaches zero. But in some languages, such as C, the programmer must manually increment and decrement the reference count when they start and stop using an object. How is this different from using manual memory allocation using malloc() and free()? [2 points]

   (ii) Some languages, such as Java, use mark-and-sweep garbage collection to manage data in heap memory. Name an advantage and a disadvantage of this approach relative to reference counting. [2 points]

2. The stack [6 points]

a. List the main contents of a stack frame, and their purposes. You may find a diagram helpful. [2 points]

b. How does a stack canary protect against run-time attacks? When should its validity be checked? [2 points]

c. Describe an attack leading to a control flow violation that is prevented by a shadow stack, but not by a stack canary. [2 points]

3. Hardware mechanisms [6 points]

a. Branch Target Integrity (BTI) checks that the target of an indirect branch (that is, a jump to a value given in a register) can only jump to a valid target.

   (i) Why is this check not made for direct branches (those whose target is encoded into the branch instruction)? [1 point]

   (ii) In a system with a hardware shadow stack, is there any gain from making a BTI check on function return addresses? [1 point]

b. What security risk follows from using the same modifier with all Pointer Authentication instructions in a program? [2 point]

c. ARM MTE attaches a (variable) colour to each region of memory. Give an example of a situation where the colour of a region of heap memory might be changed. [1 point]

d. Why do hardware capability mechanisms (like CHERI) use tagged memory? [1 point]

3. Miscellaneous topics [4 points]

a. How do subtractive attacks against watermarks differ from distorting attacks? [1 point]

b. In compilers, what is an intermediate representation (IR) and what is its purpose? [2 points]

c. Register spilling happens when the compiler runs out of registers to hold the data currently in use. Why might this be dangerous? [1 point]

4. Memory Protection CFI analysis [10 points]

We will analyse the indirect call CFI, as seen in / provided by the assembler dump produced by compiled C-code (with CFI) in the attachment (and only that, for coherence).

The C code consists of two parts: in foo.c, we define various functions, and a function get_implementation() that takes an integer argument returns a pointer to one of these four functions.

Then, in main.c, the program calls get_implementation(), which selects an implementation based on the number of command-line parameters provided to the program (passed to main() in the argc argument).

To answer this question, you will need to look at the program's assembly code for the ARMA64 platform (such as in e.g. the Macbook Air M1), provided in assembly.S. Assembler for the ARMA64 and RISC-V processors is different from that of X86, but generally simpler and easier to follow. An introduction to the ARMA64 can be found in https://modexp.wordpress.com/2018/10/30/arm64-assembly/#set, or https://courses.cs.washington.edu/courses/cse469/19wi/arm64.pdf .

Observe the **main function at address 400640 in assembly.S**., and answer the following questions.

a. The program checks the values of several variables. Using the ARM assembly code reference above, write an equation describing each of the following checks:

   (i) In get_implementation(), the value of the argument is checked on lines 188-190. [2 points]

   (ii) After main() calls get_implementation() on line 204 (address 400648), the result is returned in register x0. Then, checks are made on the return value, on lines 206-209, 213, and 217 (this code is interspersed with other instructions for reasons of optimization). [2 points]

   A hint: **adrp x10, #0** returns the address of the start of the memory page (4KB resolution) containing the code at the current PC.

b. In a(ii), you found a range of possible addresses that get_implementation() should return. After checking the return value, main() then jumps to it, on line 218. What does the code at the possible target addresses do? [2 points]

c. What does the program do if get_implementation() returns an address that fails the check from a(ii)? [2 points]

d. Why can't get_implementation() return a pointer directly to each implementation? [2 points]